On the Adaptive Security of Graph-based Games

by

Karen Klein

September, 2021

A thesis submitted to the
Graduate School
of the
Institute of Science and Technology Austria
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Committee in charge:

Uli Wagner, Chair
Krzysztof Pietrzak
Daniel Wichs
Dimitar Jetchev

Institute of Science and Technology

The thesis of Karen Klein, titled *On the Adaptive Security of Graph-based Games*, is approved by:

Supervisor: Krzysztof Pietrzak, IST Austria, Klosterneuburg, Austria

Signature: _____

Committee Member: Daniel Wichs, Northeastern University, Boston, USA, and NTT Research, Palo Alto, USA

Signature: _____

Committee Member: Dimitar Jetchev, Inpher, Lausanne, Switzerland

Signature: _____

Defense Chair: Uli Wagner, IST Austria, Klosterneuburg, Austria

Signature: _____

Signed page is on file

IST Austria Thesis, ISSN: 2663-337X

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

Signature: _____

Karen Klein
September, 2021

Signed page is on file

Abstract

Many security definitions come in two flavors: a stronger "adaptive" flavor, where the adversary can arbitrarily make various choices during the course of the attack, and a weaker "selective" flavor where the adversary must commit to some or all of their choices a-priori. For example, in the context of *identity-based encryption*, selective security requires the adversary to decide on the identity of the attacked party at the very beginning of the game whereas adaptive security allows the attacker to first see the master public key and some secret keys before making this choice. Often, it appears to be much easier to achieve selective security than it is to achieve adaptive security.

A series of several recent works shows how to cleverly achieve adaptive security in several such scenarios including generalized selective decryption [Pan07][FJP15], constrained PRFs [FKPR14], and Yao's garbled circuits [JW16]. Although the above works expressed vague intuition that they share a common technique, the connection was never made precise. In this work we present a new framework (published at Crypto '17 [JKK+17a]) that connects all of these works and allows us to present them in a unified and simplified fashion.

Having the framework in place, we show how to achieve adaptive security for proxy re-encryption schemes (published at PKC '19 [FKKP19]) and provide the first adaptive security proofs for continuous group key agreement protocols (published at S&P '21 [KPW+21]). Questioning optimality of our framework, we then show that currently used proof techniques cannot lead to significantly better security guarantees for "graph-building" games (published at TCC '21 [KKPW21a]). These games cover generalized selective decryption, as well as the security of prominent constructions for constrained PRFs, continuous group key agreement, and proxy re-encryption. Finally, we revisit the adaptive security of Yao's garbled circuits and extend the analysis of Jafargholi and Wichs in two directions: While they prove adaptive security only for a modified construction with increased online complexity, we provide the first positive results for the original construction by Yao (published at TCC '21 [KKP21a]). On the negative side, we prove that the results of Jafargholi and Wichs are essentially optimal by showing that no black-box reduction can provide a significantly better security bound (published at Crypto '21 [KKPW21c]).

Acknowledgements

I want to thank my supervisor Krzysztof Pietrzak for the great experience of doing phd studies at IST Austria. I really enjoyed the good working atmosphere within his research group and felt well supported. Krzysztof was always open for discussions and encouraged interaction within the group, which I appreciated a lot. Furthermore I want to thank Daniel Wichs for the great discussions we had on Yao's garbling scheme, which led to the final chapter of this thesis. I also want to thank Dimitar Jetchev for the inspiring discussions in the very beginning of my phd studies.

I want to thank my colleagues and co-authors for the good collaboration and the many discussions we had. Here I especially want to thank Chethan Kamath, who accompanied me throughout my entire phd studies, was always accessible for discussions, gave me great advice and at all times kept the big picture in mind. I also want to highlight the valuable collaboration with Michael Walter, whose very structured way of working and approaching research questions I appreciated a lot.

I want to thank my partner, my family and friends for their personal support, not only during my phd studies but also on my way there.

Finally, I want to acknowledge the funding by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (682815 - TOCNeT).

About the Author

Karen Klein completed a BSc and MSc in Mathematics at the University of Vienna. During her studies she did an internship at the Vienna Institute of Demography. Besides Mathematics, she also completed a MA at the University of Music and Performing Arts Vienna. In 2016, she joined IST Austria for PhD studies in the research group of Krzysztof Pietrzak. Her main research interests are applications of algebraic and combinatorial methods to theoretical computer science, and range from foundations of cryptography to practical schemes.

Beside her main research topic related to the adaptive security of graph-based games, she also contributed to various project unrelated to this thesis:

Continuous group key agreement

- Joel Alwen, Benedikt Auerbach, Mirza Ahad Baig, Miguel Cueto-Noval, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, and Michael Walter. Grafting key-trees: Efficient key-management for overlapping groups. TCC 2021, to appear, 2021

Proofs of sequential work and blockchain applications

- Hamza Abusalah, Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. Reversible proofs of sequential work. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 277–291. Springer, Heidelberg, May 2019

Automated contact tracing

- Benedikt Auerbach, Suvradip Chakraborty, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, Michael Walter, and Michelle Yeo. Inverse-sybil attacks in automated contact tracing. In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 399–421. Springer, Heidelberg, May 2021

Memory-hard functions

- Joël Alwen, Peter Gazi, Chethan Kamath, Karen Klein, Georg Osang, Krzysztof Pietrzak, Leonid Reyzin, Michal Rolinek, and Michal Rybár. On the memory-hardness of data-independent password-hashing functions. In Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim, editors, *ASIACCS 18*, pages 51–65. ACM Press, April 2018

List of Collaborators and Publications

This thesis is based on the following publications and their respective full versions. All of them were conducted in collaboration with co-authors, the contrubution of Karen Klein is enclosed in brackets, respectively.

1. Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Daniel Wichs. Limits on the adaptive security of yao's garbling. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 486–515, Virtual Event, August 2021. Springer, Heidelberg. Full version: [KKPW21d]

   [significant contribution on entire paper, most technical details carried out by herself]

2. Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. On treewidth, separators and yao's garbling. TCC 2021, to appear, 2021. Full version: [KKP21b]

   [significant contribution on development of techniques]

3. Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. The cost of adaptivity in security games on graphs. TCC 2021, to appear, 2021. Full version: [KKPW21b]

   [significant contribution on entire paper, many proofs carried out by herself]

4. Karen Klein, Guillermo Pascual Perez, Michael Walter, Chethan Kamath, Margarita Capretto, Miguel Cueto, Ilia Markov, Michelle Yeo, Joel Alwen, and Krzysztof Pietrzak. Keep the dirt: Tainted treekem, adaptively and actively secure continuous group key agreement. In *2021 IEEE Symposium on Security and Privacy*, pages 268–284. IEEE Computer Society Press, May 2021. Full version: [ACC+19]

   [main responsible for security analysis together with M. Walter and C. Kamath; only parts of the paper are included in this thesis]

5. Georg Fuchsbauer, Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. Adaptively secure proxy re-encryption. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 317–346. Springer, Heidelberg, April 2019. Full version: [FKKP18]

   [joint discussions on main body, analysis of lattice-based schemes carried out by herself]

6. Zahra Jafargholi, Chethan Kamath, Karen Klein, Ilan Komargodski, Krzysztof Pietrzak, and Daniel Wichs. Be adaptive, avoid overcommitting. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 133–163. Springer, Heidelberg, August 2017. Full version: [JKK+17b]

   [minor contribution on the framework and proof of the main theorem; only parts of the paper are included in this thesis]

Table of Contents

List of Figures

List of Tables

List of Algorithms

Part I

Introduction and Preliminaries

CHAPTER 1

Introduction

Consider the following game[1] played between a challenger C and an adversary A using a symmetric encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$. The challenger first samples, independently and uniformly at random, N keys $\mathtt{k}_1, \ldots, \mathtt{k}_N$. These correspond to users $\mathsf{ID}_1, \ldots, \mathsf{ID}_N$ respectively. The adversary A is now allowed to *adaptively* make two types of queries:

1. Ask for an encryption of $\mathtt{k}_j$ under the key $\mathtt{k}_i$ to obtain $\mathtt{c}_{i,j} \leftarrow \mathsf{Enc}_{\mathtt{k}_i}(\mathtt{k}_j)$.[2]
2. Corrupt a user U_i to obtain the key $\mathtt{k}_i$.

At the end of the game, A challenges C on a user ID_{i^*} and is given either the real key $\mathtt{k}_{i^*}$ or an independent, random key $\mathtt{r}$. A wins this "real or random game" if it correctly guesses which of the two it got. If no efficient A can win with probability higher than $1/2 + \epsilon$ we say the protocol is 2ϵ secure.

The above game can be thought of as the adversary A adaptively building a "key-graph" $G = (\mathcal{V}, \mathcal{E})$, where the vertices $\mathcal{V} = \{1, \ldots, N\}$ correspond to the users and their keys, whereas the (directed) edges $\mathcal{E}$ correspond to the encryption queries that A makes: a directed edge (i, j) is added to $\mathcal{E}$ if A requests the encryption of $\mathtt{k}_j$ under the key $\mathtt{k}_i$. Note that for i^* to be a non-trivial challenge, i^* must be a sink and must *not* be reachable (in the graph-theoretic sense) from any of the corrupted vertices – otherwise, A could simply decrypt the ciphertexts along the path from any corrupted node to the challenge to learn $\mathtt{k}_{i^*}$.

The above game is called *Generalized Selective Decryption* (GSD) and it captures the security of protocols for multicast encryption [Pan07] and continuous group key agreement [KPW+21]. Thus, the question one is interested in is whether the security of this game (given that the key-graph is *acyclic*) can be based on the IND-CPA security of the underlying encryption scheme.[3] For this we need to prove a computational soundness (i.e., security) theorem of the form: if the encryption scheme is ϵ-IND-CPA secure then the GSD game is ϵ'-secure for some ϵ' that depends on ϵ. Ideally, the loss in security should be kept to a polynomial, i.e.

[1] This introduction is taken essentially from [KKPW21a]. © IACR 2021, to appear.

[2] If A repeatedly queries encryptions of $\mathtt{k}_j$ under $\mathtt{k}_i$, it always obtains the same encryption $\mathtt{c}_{i,j}$.

[3] In case the key-graph contains cycles, one must additionally assume that the encryption scheme is key-dependent message (KDM) secure [BRS03]. Such problems are of a different flavour and we don't deal with them. As mentioned before, the GSD game is typically used to capture the security of protocols, and then the acyclicity is enforced by the protocol rules.

$\epsilon' = \epsilon \cdot \text{poly}(N)$. Otherwise, this requires to set the security parameter of the underlying encryption scheme very large, which implies very long keys and leads to inefficient schemes.

The simpler task of proving such a soundness theorem in case the adversary is *selective*, in the sense that it commits to its queries (and thus the key-graph G) at the beginning of the GSD game, is relatively straightforward to achieve. First consider the case $N = 2$. In this case, the adversay makes at most one encryption query. Clearly, if it doesn't make any encryption query, then the challenge key is perfectly indistinguishable from a random, independent key; hence A has advantage 0. Now, for a single encryption query (i, i^*), GSD-security can directly be based on the IND-CPA security of the encryption scheme by a reduction which samples two random, independent keys $\mathrm{k}_{i^*}, \mathrm{k}'_{i^*}$, sends them to the IND-CPA challenger, and answers A's encryption query by forwarding the challenge ciphertext it received. A's challenge query i^* is answered by k_{i^*}. This reduction perfectly simulates the real or random GSD game, depending on the IND-CPA challenge.

For $N > 2$, one can use a *hybrid argument*: The idea here is to define a sequence of slightly modified games "interpolating" between the real and the random GSD game, such that each pair of subsequent hybrid games in this sequence can be proven indistinguishable based on the IND-CPA security of the encryption scheme (similar to the case of $N = 2$ above). Security of selective GSD then follows at a loss linear in the number τ of hybrids: If the encryption scheme is ϵ-IND-CPA secure then the selective GSD game is $\tau\epsilon$-secure. To define such a sequence of indistinguishable hybrid games, a first attempt would be to simply switch the edges incident on the challenge node i^*, one at a time, from encryptions of k_{i^*} to encryptions of a random, independent key k'_{i^*}. However, the issue here is that a reduction can only embed a challenge at an edge (i, i^*) if i is a source node, since otherwise – to properly simulate the hybrid game – it has to output an encryption of k_i. To avoid this problem, one has to switch all encryptions associated with edges incident on node i to encryptions of a random, independent key k'_i first. Thus, one ends up with a sequence of hybrids, where – starting from edges outgoing from source nodes in the key-graph – for all edges in the graph the associated ciphertexts are switched, one at a time, from real encryptions of the respective sink key k_i to encryptions of a random, independent key k'_i. Once all edges incident on the challenge node are switched, one can then switch back all edges not incident on the challenge, in reverse order. Since there are at most $N(N-1)/2$ edges in the key-graph (directed acyclic graph), this leads to a sequence of indistinguishable hybrids of length $\tau = N(N-1)$, and hence implies $N(N-1) \cdot \epsilon$-security of selective GSD.

The study of *adaptive* security of GSD, where the key-graph is unknown at the beginning of the game and is only gradually revealed during the query phase, was initiated by Panjwani in [Pan07] and remains notoriously hard. The difficulty here arises from the fact that the reduction when answering A's encryption queries has to guess their position in the final key-graph. Non-trivial results for adaptive GSD are only known in settings where the adversary is restricted to specific key-graphs (which needs to be enforced by the higher level protocol).

The goal of this thesis is to shed a little more light on the problem of proving adaptive security, not only for GSD but also for other security games that involve some graph structure.

1.1 Our Contributions

1.1.1 A Framework for Adaptive Security

As a first step[4], in Chapter 3 we present a framework for proving adaptive security, which we refer to as *Piecewise-Guessing framework*[5]. This framework connects the known results on GSD from [Pan07, FJP15] with techniques used in the context of constrained pseudorandom functions (cPRF) [FKPR14] and garbled circuits [JW16], and allows us to present all these works in a unified and simplified way.

Underlying our framework is the following simple idea. It is well known that selective security, where the adversary A commits to n bits of information about their future choices, automatically implies adaptive security at the cost of amplifying A's advantage by a factor of up to 2^n. However, in some cases (e.g. GSD above) the proof of selective security proceeds via a sequence of hybrids, where each pair of adjacent hybrids locally only requires some smaller partial information consisting of $m \ll n$ bits. The partial information needed might be completely different between different pairs of hybrids, and if we look across all the hybrids we might rely on the entire n-bit commitment. Nevertheless, the above is sufficient to prove adaptive security, at the cost of amplifying the adversary's advantage by a factor of only $2^m \ll 2^n$.

As examples using the above framework we consider the known results on GSD [Pan07, FJP15], cPRFs [FKPR14], and garbled circuits [JW16]. For all these examples the different hybrids can be captured by some sort of a *graph pebbling game* and the amount of information that the adversary needs to commit to in each pair of hybrids is related to the maximum number of pebbles in play at any point in time. Therefore, coming up with better strategies for proving adaptive security translates to various pebbling strategies for different types of graphs.

1.1.2 Adaptively Secure Proxy Re-encryption

As a new application of the Piecewise-Guessing framework[6], in Chapter 4 we analyze the adaptive security of proxy re-encryption (PRE) schemes. A PRE scheme is a public-key encryption scheme that allows the holder of a key pair $(\mathtt{pk}, \mathtt{sk})$ to derive a re-encryption key for another public key $\mathtt{pk}'$. This re-encryption key lets anyone transform ciphertexts under $\mathtt{pk}$ into ciphertexts under $\mathtt{pk}'$ without having to know the underlying message, while transformations from $\mathtt{pk}'$ to $\mathtt{pk}$ should not be possible (unidirectional). Security is defined in a multi-user setting against an adversary that gets the users' public keys and can ask for re-encryption keys and can corrupt users by requesting their secret keys. Any ciphertext that the adversary cannot trivially decrypt given the obtained secret and re-encryption keys should be secure.

All existing security proofs for PRE only show *selective* security, where the adversary must first declare the users it wants to corrupt. This can be lifted to more meaningful *adaptive* security by guessing the set of corrupted users among the N users, which loses a factor exponential in N, rendering the result meaningless already for moderate N.

[4] This abstract is taken essentially from [JKK+17a]. https://doi.org/10.1007/978-3-319-63688-7_5.

[5] This name was introduced by Kowalczyk and Wee [KW19], who used our framework to obtain adaptively secure attribute-based encryption.

[6] This abstract is taken essentially from [FKKP19]. https://doi.org/10.1007/978-3-030-17259-6_11.

To avoid the exponential loss that results from guessing the adaptive choices made by an adversary, we apply the Piecewise-Guessing framework to PRE schemes that satisfy some natural additional properties. Concretely, we give a more fine-grained reduction for several unidirectional PRE schemes, proving adaptive security at a much smaller loss. The loss depends on the graph of users whose edges represent the re-encryption keys queried by the adversary. For trees and chains the loss is quasi-polynomial in the size and for general graphs it is exponential in their depth and indegree (instead of their size as for previous reductions). Fortunately, trees and low-depth graphs cover many, if not most, interesting applications.

Our results apply e.g. to the bilinear-map based PRE schemes by Ateniese et al. [AFGH05, ABH09], Gentry's FHE-based scheme [Gen09] and the LWE-based scheme by Chandran et al. [CCL+14].

1.1.3 Adaptively Secure Continuous Group Key Agreement

In Chapter 5 we present[7] an application of the Piecewise-Guessing framework to continuous group key agreement (CGKA) for secure messaging within dynamically changing groups. While messaging systems with strong security guarantees are widely used in practice, designing a protocol that scales efficiently to large groups and enjoys similar security guarantees remained largely open. The two existing proposals to date are ART [CCG+18] and TreeKEM [BBM+20]. TreeKEM is the currently considered candidate by the IETF MLS working group, but dynamic group operations (i.e. adding and removing users) can cause efficiency issues. Therefore we formalize and analyze a variant of TreeKEM which we term Tainted TreeKEM (TTKEM for short); the basic idea underlying TTKEM was suggested by Millican (MLS mailing list, February 2018[8]). This version is more efficient than TreeKEM for some natural distributions of group operations; we do not provide further details in this thesis, but refer to our publication [KPW+21], where we quantify the efficiency through simulations.

We provide two security proofs for TTKEM which establish post compromise and forward secrecy even against adaptive attackers. If M is the group size and Q the number of operations, the security loss (to the underlying PKE) in the random oracle model is a polynomial factor $(QM)^2$, and in the standard model a quasipolynomial $Q^{\log(M)}$. While the latter result is derived rather straight-forward from the Piecewise-Guessing framework (similar to the case of GSD), for the former polynomial bound we develop a new result for a public-key version of GSD in the random oracle model, which might be of independent interest. Both our proofs can be adapted to TreeKEM as well. Before our work no security proof for any TreeKEM-like protocol establishing meaningful security guarantees against an adversary who can *adaptively* choose the sequence of operations was known. We also are the first to prove (or even formalize) *partially active* security where the server can arbitrarily deviate from the protocol specification. Proving fully active security – where also the users can arbitrarily deviate – remains open.

1.1.4 Adaptive Indistinguishability of Yao's Garbling

In Chapter 6, we revisit[9] the adaptive security of Yao's garbled circuits [Yao86]. Given a circuit C and an input x, a garbling scheme allows to output a garbled circuit $\widetilde{\mathsf{C}}$ and a garbled input

[7] This abstract is taken essentially from [KPW+21]. © 2021 IEEE, https://doi.org/10.1109/SP40001.2021.00035.

[8] [MLS] Removing members from groups, Jon Millican {jmillican@fb.com}, 12 February 2018, https://mailarchive.ietf.org/arch/msg/mls/4-gvXpc-LGbWoUS7DKGYG65lkxs

[9] This abstract is taken essentially from [KKP21a]. © IACR 2021, to appear.

$\widetilde{x}$ that only reveal the output $\mathsf{C}(x)$ while everything else – besides some leakage such as the size or topology of the circuit – remains hidden. In many applications one requires security of the scheme also in the adaptive setting, where the adversary first – in an *offline* phase – chooses a circuit C and then (after receiving its garbling) – in the *online* phase – adaptively chooses its input x. Here, for the sake of efficiency, the cost during the online phase is to be kept minimal.

Lindell and Pinkas [LP09] gave a formal proof of security in the *selective* setting assuming secure symmetric-key encryption (and hence one-way functions). This was followed by results, both positive and negative, concerning its security in the, stronger, *adaptive* setting: Applebaum et al. [AIKW13] showed that Yao's garbling scheme cannot satisfy adaptive security as is, due to a simple incompressibility argument. Jafargholi and Wichs [JW16] considered a natural adaptation of the scheme that circumvents this negative result, and proved that it is adaptively secure, at least for shallow circuits. In particular, they showed that for the class of circuits of depth D, the loss in security is at most exponential in D. The above results all concern the *simulation-based* notion of security.

We show that Yao's garbling scheme is adaptively *indistinguishable* – a weaker security notion than the more common simulation-based one – for the class of Boolean circuits of size N and treewidth w with only an $N^{O(w)}$ loss in security. For instance, circuits with constant treewidth are as a result adaptively indistinguishable with only a polynomial loss. This (partially) complements the negative result of Applebaum et al. [AIKW13]. As main technical contributions, we introduce a new pebble game that abstracts out our security reduction and then present a pebbling strategy for this game where the number of pebbles used is roughly $O(\delta_{out} w \log(N))$, δ_{out} being the fan-out of the circuit. The design of the strategy relies on separators, a graph-theoretic notion with connections to circuit complexity.

1.1.5 On the Cost of Adaptivity in Security Games on Graphs

In many of the applications of the Piecewise-Guessing framework[10] (GSD, cPRF, PRE, CGKA) an adversary adaptively reveals edges of some graph and the loss in security is involved by guessing certain edges in the final graph. In Chapter 7 we initiate the study of *lower bounds* on the loss in adaptive security for these cryptographic protocols. We prove lower bounds that almost match the upper bounds (proven using the Piecewise-Guessing framework).

Some of our lower bounds only apply to a restricted class of black-box reductions which we term "oblivious" (the existing upper bounds are of this restricted type), some apply to the broader but still restricted class of non-rewinding reductions, while our lower bound for proxy re-encryption applies to all black-box reductions. The fact that some of our lower bounds seem to crucially rely on obliviousness or at least a non-rewinding reduction hints to the exciting possibility that the existing upper bounds can be improved by using more sophisticated reductions.

Our main conceptual contribution is a two-player multi-stage game called the Builder-Pebbler Game. We translate bounds on the winning probabilities for various instantiations of this game into cryptographic lower bounds for the above-mentioned primitives using oracle separation techniques.

[10] This abstract is taken essentially from [KKPW21a]. © IACR 2021, to appear.

1.1.6 Limits on the Adaptive Security of Yao's Garbling

In Chapter 8, we also analyze[11] the adaptive security of Yao's garbling scheme from a negative side and show that the upper bound of Jafargholi and Wichs is basically optimal in a strong sense. As our main result, we show that there exists a family of Boolean circuits, one for each depth $D \in \mathbb{N}$, such that *any* black-box reduction proving the adaptive *indistinguishability-security* of the natural adaptation of Yao's scheme from any symmetric-key encryption has to lose a factor that is exponential in $\sqrt{D}$. Since indistinguishability is a weaker notion than simulatability, our bound also applies to adaptive simulation-based security.

To establish our results, we build on the approach from Chapter 7, which uses pebbling lower bounds in conjunction with oracle separations to prove fine-grained lower bounds on the loss in cryptographic security. While the difficulty for the reduction (playing the role of the pebbler) in "graph-building" games as discussed there lies in the fact that the graph is only revealed edge-by-edge, in the setting of garbled circuits, in contrast, the graph structure is initially known and the game has just two rounds. The difficulty of the reduction here comes from having to guess the bits running over a subset of wires during evaluation of the circuit.

Epilogue

In this thesis we put forth the research on adaptive security of graph-based games. While we do achieve several interesting positive as well as negative results, we still leave open many exciting questions. For multi-round games such as GSD or PRE, where the adversary adaptively queries edges and the reduction needs to guess the position of these edges in the final graph, we prove that the quasi-polynomial upper bounds from [Pan07, FKPR14] and Chapter 4 for restricted graph structures (paths, binary trees) are essentially optimal when considering *oblivious* black-box reductions. However, it is not clear at this point how non-obliviousness could be exploited and whether this could potentially lead to polynomial bounds for these restricted classes of graphs. For unrestricted graph structures, we prove that no *non-rewinding* black-box reduction can achieve a polynomial loss; but again, similar to the case of obliviousness, it is not clear how rewinding could be used to prove stronger upper bounds. Only for PRE on unrestricted graphs and for Yao's garbling we are able to prove lower bounds for *arbitrary* black-box reductions; but also here the picture is not yet complete and we leave it for future research to construct actual counter examples.

[11] This abstract is taken essentially from [KKPW21c]. https://doi.org/10.1007/978-3-030-84245-1_17.

CHAPTER 2

Preliminaries

2.1 Notation and Basic Definitions

2.1.1 General notation

Throughout, we use λ to denote the security parameter. For integers $a, b \in \mathbb{Z}$ with $a \leq b$, by $[a, b]$ we denote the set $\{a, a+1, \ldots, b\}$. For two strings r, s we write $r||s$ to denote their concatenation, and $s[i]$ for the ith entry. For a string $s = s_1, \ldots, s_n$, let $|s| := n$ denote its length. For $1 \leq a < b \leq n$ we define $s[a, b] := s_a, \ldots, s_b$ and $s[0] := \emptyset$ the empty string.

Abusing notation we also use $[0, 1]$ for the closed interval of real numbers $r \in \mathbb{R}$ with $0 \leq r \leq 1$. A function $\epsilon : \mathbb{N} \to [0, 1]$ is *negligible* if for every polynomial $p(\lambda)$ there exists a $\lambda_0 \in \mathbb{N}$ such that $\epsilon(\lambda) < 1/p(\lambda)$ for all $\lambda \geq \lambda_0$. We will only consider logarithms to the base 2 (i.e., $\log := \log_2$). We use calligraphic letters like $\mathcal{X}$ to denote sets and sans-serif letters like X to denote algorithms.

2.1.2 Graphs

In this thesis we only consider (directed and undirected) *simple* graphs, i.e. graphs without loops or multiple edges. For $N = N(\lambda)$, $G = (\mathcal{V}, \mathcal{E})$ denotes a directed acyclic graph (DAG) with $\mathcal{V} = \{v_1, \ldots, v_N\}$ and $\mathcal{E} \subseteq \mathcal{V}^2$. Often we associate vertices v_i with the associated index i. The set of all graphs of N vertices is denoted by $\mathcal{G}(N)$. For a subset of nodes $\mathcal{X} \subseteq \mathcal{V}$ we denote by $G_{|\mathcal{X}}$ the subgraph of G induced on the set of vertices in $\mathcal{X}$. That is $G_{|\mathcal{X}} = (\mathcal{X}, \mathcal{E}_{|\mathcal{X}})$ where $\mathcal{E}_{|\mathcal{X}} := \{(u, v) \in \mathcal{E} : u, v \in \mathcal{X}\}$.

For $v \in \mathcal{V}$, we define the parents (or predecessors) of v as $\mathsf{parents}(v) := \{u : (u, v) \in \mathcal{E}\}$ and the set of ingoing edges as $\mathsf{in}(v) := \{(u, v) : u \in \mathsf{parents}(v)\}$. Similarly, we define the children (or successors) of v as $\mathsf{children}(v) := \{u : (v, u) \in \mathcal{E}\}$. Sometimes we make the graph explicit by writing $\mathsf{parents}_G(v)$ and $\mathsf{children}_G(v)$. These definitions can naturally be extended to a set of vertices $\mathcal{X} \subseteq \mathcal{V}$ as $\mathsf{in}(\mathcal{X}) := \bigcup_{v \in \mathcal{X}} \mathsf{in}(v)$, as well as $\mathsf{parents}(\mathcal{X}) := \bigcup_{v \in \mathcal{X}} \mathsf{parents}(v)$ and $\mathsf{children}(\mathcal{X}) := \bigcup_{v \in \mathcal{X}} \mathsf{children}(v)$. Finally, we say that u is adjacent to v if it is either a predecessor or a successor of v.

The degree of a vertex v is the number of nodes that are adjacent to v. The indegree (resp., outdegree) of a vertex is defined as the number of parents, i.e. the number of edges coming in to (resp., going out of) that vertex. The degree δ (resp., indegree δ_{in}, outdegree δ_{out}) of the

graph is the maximum degree (resp., indegree, outdegree) over all the vertices. A vertex with indegree (resp., outdegree) zero is called a source (resp., sink). A vertex u is connected to another vertex v (or alternatively v is reachable from u) if there is a directed path from u to v in G. The depth D of G is the length of the longest path in G, where the length of a path is the number of edges in the path.

For a set of m edges $\mathcal{P} = \{(v_i, w_i)\}_{i=1}^m \subseteq \mathcal{E}$, let $\mathcal{V}(\mathcal{P}) := \bigcup_{i=1}^m \{v_i, w_i\}$ denote the set of nodes that have an incident edge in $\mathcal{P}$. The edge set $\mathcal{P}$ is called *disjoint*, if they do not share a node, i.e. if $|\mathcal{V}(\mathcal{P})| = |\bigcup_{i=1}^m \{v_i, w_i\}| = 2m$.

Sometimes we assume that the set of edges $\mathcal{E}$ is (totally) ordered: we use $(u, v) < (u', v')$ to denote that (u, v) precedes (u', v') in the set. We assume that parents$(\cdot)$ preserves the order on $\mathcal{E}$ — i.e., if $(u, v) < (u', v)$ then u precedes u' in parents(v). Finally, parents$^{-1}(v)$ denotes the set parents(v) with elements in reverse order.

2.1.3 Sets, distributions, algorithms

For two sets $\mathcal{X}, \mathcal{Y}$ we write $\mathcal{X} \Delta \mathcal{Y} := (\mathcal{X} \setminus \mathcal{Y}) \cup (\mathcal{Y} \setminus \mathcal{X})$ for the symmetric difference. We write $x \leftarrow \mathcal{X}$ for sampling an element x uniformly at random from the set $\mathcal{X}$; analogously, $x_1, \ldots, x_N \leftarrow \mathcal{X}$ denotes sampling $x_1, \ldots, x_N$ independently and uniformly at random from the set $\mathcal{X}$. We use $U_{\mathcal{X}}$ to denote the uniform distribution over $\mathcal{X}$ and U_n to denote the uniform distribution over $\{0,1\}^n$.

For two variables X, Y, we write $X \sim Y$ to denote that X and Y have the same distributions and $\Delta(X, Y)$ to denote their statistical distance. To indicate sampling according to a distribution X on $\mathcal{X}$, we write $x \leftarrow X$. By $[X]$ we denote the support of X, i.e., the values with positive probability.

Two distributions $X = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda \in \mathbb{N}}$ are (t, ϵ)-indistinguishable (with $t = t(\lambda), \epsilon = \epsilon(\lambda)$), denoted $X \approx_{(t,\epsilon)} Y$, if for every adversary A running in time at most t

$$|\Pr[\mathsf{A}(X) = 1] - \Pr[\mathsf{A}(Y) = 1]| \leq \epsilon.$$

Throughout the paper, we will repeatedly use the following lemma concerning the transitivity of the indistinguishability relation $\approx$:

Lemma 1. *Let X, Y, Z be distributions on a set $\mathcal{X}$. If $X \approx_{(t_1,\epsilon_1)} Y$ and $Y \approx_{(t_2,\epsilon_2)} Z$, then $X \approx_{(\min(t_1,t_2),\epsilon_1+\epsilon_2)} Z$.*

For an algorithm A, we use t_{A} to denote its run time; in a similar manner, for a set $\mathcal{X}$, we use $t_{\mathcal{X}}$ to denote the complexity of sampling from $\mathcal{X}$ uniformly at random. For $\mathcal{X} = \{0,1\}^n$, we use t_n to denote the time complexity of sampling a string of length n uniformly at random. For two algorithms A, B, we denote by $\mathsf{A} \equiv \mathsf{B}$ that A has exactly the same input/output distribution as B.

2.2 IND-CPA Secure Encryption

Throughout this thesis we will repeatedly refer to the notion of IND-CPA secure encryption, in the *secret-key* setting as well as in the *public-key* setting. The definitions are taken from [KL14].

Symmetric-key encryption (SKE)

Definition 1. A *secret-key encryption (SKE)* scheme with key space $\mathcal{K}$, message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ is a tuple of probabilistic polynomial-time (PPT) algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that

- $\mathsf{Gen}(1^\lambda)$: The *key-generation* algorithm Gen takes as input a security parameter 1^λ (in unary) and outputs a key $\mathtt{k} \in \mathcal{K}$.
- $\mathsf{Enc}_\mathtt{k}(\mathtt{m})$: The *encryption* algorithm Enc takes as input a key $\mathtt{k} \in \mathcal{K}$ and a message $\mathtt{m} \in \mathcal{M}$, and outputs a ciphertext $\mathtt{c} \in \mathcal{C}$.
- $\mathsf{Dec}_\mathtt{k}(\mathtt{c})$: The *decryption* algorithm Dec takes as input a key $\mathtt{k} \in \mathcal{K}$ and a ciphertext $\mathtt{c} \in \mathcal{C}$, and outputs a message $\mathtt{m} \in \mathcal{M}$ (or $\perp$).

For correctness, it is required that for all $\mathtt{k} \leftarrow \mathsf{Gen}(1^\lambda)$ and $\mathtt{m} \in \mathcal{M}$

$$\mathsf{Dec}_\mathtt{k}(\mathsf{Enc}_\mathtt{k}(\mathtt{m})) = \mathtt{m}.$$

An SKE scheme is called (t, ϵ)-*indistinguishable under chosen-plaintext attack* (IND-CPA secure) if the following games G_0 and G_1 are (t, ϵ)-indistinguishable: The game G_b ($b \in \{0, 1\}$) is played between an adversary A and a challenger C, both given input the security parameter 1^λ. First, C samples a key $\mathtt{k} \leftarrow \mathsf{Gen}(1^\lambda)$. The adversary A can then make the following types of queries:

- Encryption queries $(\mathtt{encrypt}, \mathtt{m})$ with $\mathtt{m} \in \mathcal{M}$: C returns $\mathtt{c} \leftarrow \mathsf{Enc}_\mathtt{k}(\mathtt{m})$.
- One challenge query $(\mathtt{challenge}, \mathtt{m}_0, \mathtt{m}_1)$ with $\mathtt{m}_0, \mathtt{m}_1 \in \mathcal{M}$: C returns the challenge ciphertext $\mathtt{c}^* \leftarrow \mathsf{Enc}_\mathtt{k}(\mathtt{m}_b)$.

In this work we assume $\mathcal{K} \subseteq \mathcal{M}$ (i.e. we can encrypt keys), and sometimes also $\mathcal{C} \subseteq \mathcal{M}$ (i.e. we can encrypt ciphertexts). Sometimes we neglect the key generation algorithm Gen and simply assume it samples keys uniformly at random from $\mathcal{K}$.

Public-key encryption (PKE)

Definition 2. A *public-key encryption (PKE)* scheme with public/secret key space $\mathcal{K}_{pub}, \mathcal{K}_{sec}$, message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ is a tuple of PPT algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that

- $\mathsf{Gen}(1^\lambda)$: The *key-generation* algorithm Gen takes as input a security parameter 1^λ (in unary) and outputs a key pair $(\mathtt{pk}, \mathtt{sk}) \in \mathcal{K}_{pub} \times \mathcal{K}_{sec}$.
- $\mathsf{Enc}_\mathtt{pk}(\mathtt{m})$: The *encryption* algorithm Enc takes as input a public key $\mathtt{pk} \in \mathcal{K}_{pub}$ and a message $\mathtt{m} \in \mathcal{M}$, and outputs a ciphertext $\mathtt{c} \in \mathcal{C}$.
- $\mathsf{Dec}_\mathtt{sk}(\mathtt{c})$: The *decryption* algorithm Dec takes as input a secret key $\mathtt{sk} \in \mathcal{K}_{sec}$ and a ciphertext $\mathtt{c} \in \mathcal{C}$, and outputs a message $\mathtt{m} \in \mathcal{M}$ (or $\perp$).

For correctness, it is required that for all $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and $\mathtt{m} \in \mathcal{M}$

$$\mathsf{Dec}_{\mathtt{sk}}(\mathsf{Enc}_{\mathtt{pk}}(\mathtt{m})) = \mathtt{m}.$$

A PKE scheme is called (t, ϵ)-*indistinguishable under chosen-plaintext attack* (IND-CPA secure) if the following games G_0 and G_1 are (t, ϵ)-indistinguishable: The game G_b ($b \in \{0, 1\}$) is played between an adversary A and a challenger C, both given input the security parameter 1^λ. First, C samples a key pair $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and sends the public key $\mathtt{pk}$ to A. The adversary A can then make a challenge query $(\mathtt{challenge}, \mathtt{m}_0, \mathtt{m}_1)$ with $\mathtt{m}_0, \mathtt{m}_1 \in \mathcal{M}$, upon which C returns the challenge ciphertext $\mathtt{c}^* \leftarrow \mathsf{Enc}_{\mathtt{pk}}(\mathtt{m}_b)$.

Similar to the case of SKE, we assume $\mathcal{K}_{sec} \subseteq \mathcal{M}$ (i.e. we can encrypt secret keys).

Part II

Framework

CHAPTER 3

A Framework for Adaptive Security

3.1 Introduction

A series of recent works achieves adaptive security in several scenarios where we previously only knew how to achieve selective security: *generalized selective decryption (GSD)* [Pan07, FJP15], *constrained PRFs* [FKPR14], and *garbled circuits* [JW16]. Although some of these works suggest a vague intuition that there is a general technique at play, there was no attempt to make this precise and to crystallize what the technique is or how these results are connected. In this work we present a new framework that connects all of these works and allows us to present them in a unified and simplified fashion.

At a high level, our framework carefully combines two basic tools commonly used throughout cryptography: *random guessing* (of the adaptive choices to be made by the adversary)[1] and the *hybrid argument*. Firstly, "random guessing" gives us a generic way to qualitatively upgrade selective security to adaptive security at a quantitative cost in the amount of security. In particular, assume we can prove the security of a selective game where the adversary commits to n bits of information about their future choices. Then, we can also prove adaptive security by guessing this commitment and taking a factor of 2^n loss in the security advantage. However, this quantitative loss is often too high and hence we usually wish to avoid it or at least lower it. Secondly, the hybrid argument allows us to prove the indistinguishability of two games G_L and G_R by defining a sequence of hybrid games $\mathsf{G}_\mathsf{L} \equiv \mathsf{H}_0, \mathsf{H}_1, \ldots, \mathsf{H}_\tau \equiv \mathsf{G}_\mathsf{R}$ and showing that each pair of neighbouring hybrids H_i and H_{i+1} are indistinguishable.

Our Framework. Our framework starts with two adaptive games G_L and G_R that we wish to show indistinguishable but we don't initially have any direct way of doing so. Let H_L and H_R be selective versions of the two games respectively, where the adversary initially has to commit to some information $w \in \{0,1\}^n$ about their future choices. Furthermore, assume there is some sequence of selective hybrids $\mathsf{H}_\mathsf{L} = \mathsf{H}_0, \mathsf{H}_1, \ldots, \mathsf{H}_\tau \equiv \mathsf{H}_\mathsf{R}$ such that we can show that

This Chapter essentially replicates, with permission, large parts of the full version [JKK+17b] of our publication [JKK+17a], © IACR 2017, https://doi.org/10.1007/978-3-319-63688-7_5.

[1] In many previous works – including [FJP15, FKPR14, JW16] as well as a previous version of our paper [JKK+17a] – this random guessing was referred to as "complexity leveraging", but this seems to be an abuse of the term. Instead, complexity leveraging [CGGM00] refers to the use of two different schemes, S_1, S_2, where the two schemes are chosen with different values of the security parameter, $\mathtt{k}_1$ and $\mathtt{k}_2$, where $\mathtt{k}_1 < \mathtt{k}_2$, and such that an adversary against S_2 (or perhaps even the honest user of S_2) can break the security of S_1.

H_i and H_{i+1} are indistinguishable. A naïve combination of the hybrid argument and random guessing shows that G_L and G_R are indistinguishable at a factor of $2^n \cdot \tau$ loss in security, but we want to do better.

Recall that the hybrids H_i are selective and require the adversary to commit to w. However, it might be the case that for each i we can prove that H_i and H_{i+1} would be indistinguishable even if the adversary didn't have to commit to all of w but only some partial-information $\mathsf{h}_i(w) \in \{0,1\}^m$ for $m \ll n$ (formalizing this condition precisely requires great care and is the major source of subtlety in our framework). Notice that the partial information that we need to know about w may be completely different for different pairs of hybrids, and if we look across all hybrids then we may need to know all of w. Nevertheless, we prove that this suffices to show that the adaptive games G_L and G_R are indistinguishable with only a $2^m \cdot \tau \ll 2^n \cdot \tau$ loss in security. In the following chapters we will refer to our framework as *Piecewise-Guessing framework* – this name was introduced by Kowalczyk and Wee in [KW19], who used our framework in the setting of attribute-based encryption.

Applications of Our Framework. We show how to understand all of the prior works mentioned above as applications of our framework. In many cases, this vastly simplifies prior works.

In all of the examples, we get a series of selective hybrids $\mathsf{H}_1, \ldots, \mathsf{H}_\tau$ that correspond to *pebbling configurations* in some graph pebbling game. The amount of information needed to show that neighbouring hybrids H_i and H_{i+1} are indistinguishable only depends on the configuration of the pebbles in the i'th step of the game. Therefore, using our framework, we translate the problem of coming up with adaptive security proofs to the problem of coming up with pebbling strategies that only require a succinct representation of each pebbling configuration.

We now proceed to give a high level overview of our results applying our general framework to GSD, constrained PRFs, and Yao's garbled circuits, and refer to Sections 3.3 and 3.4, as well as Chapter 6 for technical details.

3.1.1 Generalized Selective Decryption

Generalized Selective Decryption (GSD) was already briefly discussed in the introduction 1: This game was introduced by Panjwani [Pan07] in order to capture the difficulty of proving adaptive security of certain protocols, most notably (a variant of) the Logical Key Hierarchy (LKH) [WGL00] multicast encryption protocol. On a high level, it deals with the scenario where we have many secret keys $\mathtt{k}_i$ and various ciphertexts encrypting one key under another (but no cycles). We will discuss this problem in depth in Section 3.3, here giving a high level overview on how our framework applies to this problem.

Let $(\mathsf{Enc}, \mathsf{Dec})$ be an IND-CPA secure symmetric encryption scheme with (probabilistic) $\mathsf{Enc}\colon \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ and $\mathsf{Dec}\colon \mathcal{K} \times \mathcal{C} \to \mathcal{M}$. We assume $\mathcal{K} \subseteq \mathcal{M}$, i.e., we can encrypt keys. In the game, the challenger – either G_L or G_R – picks $N+1$ independent uniformly random keys $\mathtt{k}_0, \ldots, \mathtt{k}_N \in \mathcal{K}$, and the adversary A is then allowed to make three types of queries:

- *Encryption queries*: on input $(\mathtt{encrypt}, i, j)$ receives $\mathsf{Enc}_{\mathtt{k}_i}(\mathtt{k}_j)$.
- *Corruption queries*: on input $(\mathtt{corrupt}, i)$ receives $\mathtt{k}_i$.

- *Challenge query*, only one is allowed: on input $(\texttt{challenge}, i)$ receives $\texttt{k}_i$ in the real game G_L, and an independent uniformly random key in the random game G_R.

We think of this game as generating a directed graph, with vertex set $\mathcal{V} = \{v_0, \ldots, v_N\}$, where every $(\texttt{encrypt}, i, j)$ query adds a directed edge (v_i, v_j), and we say a vertex v_i is corrupted if a query $(\texttt{corrupt}, i)$ was made, or v_i can be reached from a corrupted vertex. The goal of the adversary is to distinguish the games G_L and G_R, with the restriction that the constructed graph has no cycles, and the challenge vertex is a sink. To prove security, i.e., reduce the indistinguishability of G_L and G_R to the IND-CPA security of the SKE scheme, we can consider a selectivized version of this game where A must commit to the graph as described above (which uses $< N^2$ bits). The security of this selectivized game can then be reduced to the security of the SKE scheme by a series of $< N^2$ hybrids, where a distinguisher for any two consecutive hybrids can be used to break the security of the SKE scheme with the same advantage. Using random guessing followed by a hybrid argument we conclude that if the SKE scheme is ϵ-secure, the GSD game is $\epsilon \cdot N^2 \cdot 2^{N^2}$-secure. Thus, we lose an exponential in N^2 factor in the reduction.

Fortunately, if we look at the actual protocols that GSD is supposed to capture, it turns out that the graphs that A can generate are not totally arbitrary. Two interesting cases are given by GSD restricted to graphs of bounded depth, and to trees. For these cases better reductions exist. Panjwani [Pan07] shows that if the adversary is restricted to play the game such that the resulting graph is of depth at most D, a reduction losing a factor $(2N)^{D+1}$ exists. Moreover, Fuchsbauer et al. [FJP15] give a reduction losing a factor $N^{3 \log N}$ when the underlying graph is a tree. In Section 3.3 we prove these results in our framework.[2] Our proofs are much simpler than the original ones, especially than the proof of [Pan07] which is very long and technical. This is thanks to our modular approach, where our general framework takes care of delicate probabilistic arguments, and basically just leaves us with the task of designing pebbling strategies, where each pebbling configuration has a succinct description, for various graphs, which is a clean combinatorial problem. The generic connection between adaptive security proofs of the GSD problem and graph pebbling is entirely new to this work.

GSD on a Path. Let us sketch the proof idea for the [FJP15] result, but for an even more restricted case where the graph is a path visiting every node exactly once. In other words there is a permutation π over $\{0, \ldots, N\}$ and the adversary's queries are of the form $(\texttt{encrypt}, \pi(i-1), \pi(i))$ and $(\texttt{challenge}, \pi(N))$. We first consider the selective game where A must commit to this permutation π ahead of time. Let $\mathsf{H}_\mathsf{L}, \mathsf{H}_\mathsf{R}$ be the selectivized versions of G_L, G_R respectively.

To prove selective security, we can define a sequence of hybrid games $\mathsf{H}_\mathsf{L} = \mathsf{H}_0, \ldots, \mathsf{H}_\tau = \mathsf{H}_\mathsf{R}$. Each hybrid is defined by a path, $v_{\pi(0)} \to v_{\pi(1)} \to \ldots \to v_{\pi(N)}$, with a subset of the edges holding a black pebble. In the hybrid games, a pebble on $(v_{\pi(i)}, v_{\pi(i+1)})$ means that instead of answering the query $(\texttt{encrypt}, \pi(i), \pi(i+1))$ with the "real" answer $\mathsf{Enc}_{\texttt{k}_{\pi(i)}}(\texttt{k}_{\pi(i+1)})$, we answer it with a "fake" answer $\mathsf{Enc}_{\texttt{k}_{\pi(i)}}(\texttt{r})$ for a uniformly random key $\texttt{r}$. The goal is to move from a hybrid with no pebbles (this corresponds to H_L) to one with a single black pebble on the "sink" edge $(v_{\pi(N-1)}, v_{\pi(N)})$ (this corresponds to H_R). We can prove that neighbouring hybrids are indistinguishable via a reduction from IND-CPA security as long as the pebbling configurations are only modified via the following legal moves:

[2] In fact, we prove a slightly weaker result than [Pan07] which depends on the indegree of the resulting graph; bounding the indegree by N we obtain an additional factor 2 in the exponent.

1. We can put/remove a pebble on the source edge $(v_{\pi(0)}, v_{\pi(1)})$ at any time.
2. We can put/remove a pebble on an edge $(v_{\pi(i)}, v_{\pi(i+1)})$ if the preceding edge $(v_{\pi(i-1)}, v_{\pi(i)})$ has a pebble.

This is because adding/removing a pebble $(v_{\pi(i)}, v_{\pi(i+1)})$ means changing what we encrypt under key $\mathrm{k}_{\pi(i)}$ and therefore we need to make sure that either the edge is a source edge or there is already a pebble on the preceding edge to ensure that the key $\mathrm{k}_{\pi(i)}$ is never being encrypted under some other key.

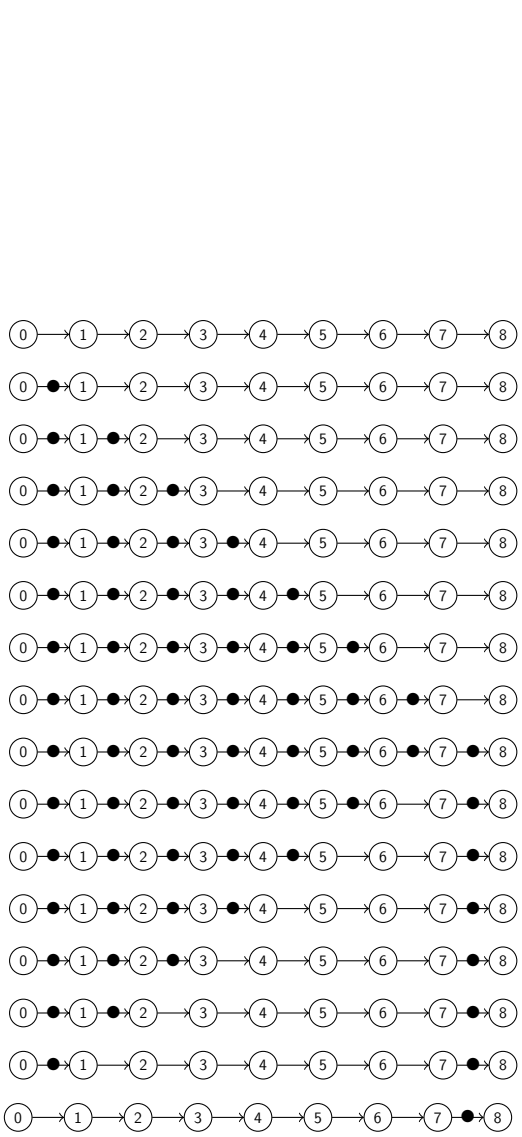
The simplest "basic pebbling strategy" consists of $2N$ moves where we add pebbles on the path $v_{\pi(0)} \to v_{\pi(1)} \to \ldots \to v_{\pi(N)}$, one by one starting on the left and then remove one by one starting on the right, keeping only the pebble on the sink edge $(v_{\pi(N-1)}, v_{\pi(N)})$. This is illustrated in Figure 3.1.(a) for $N = 8$. The strategy uses N pebbles. However, there are other pebbling strategies that allow us to trade off more moves for fewer pebbles. For example there is a "recursive strategy" (recursively pebble the middle vertex, then recursively pebble the right-most vertex, then recursively remove the pebble from the middle vertex) that uses at most $\log N + 1$ pebbles (instead of N), but requires $3^{\log N} + 1$ moves (instead of just $2N$). This is illustrated in Figure 3.1.(b).

As we described, each pebbling strategy with τ moves gives us a sequence of hybrids $\mathsf{H}_\mathsf{L} = \mathsf{H}_0, \ldots, \mathsf{H}_\tau = \mathsf{H}_\mathsf{R}$ that allows us to prove selective security. Furthermore, we can prove relatively easily that neighbouring hybrids $\mathsf{H}_j, \mathsf{H}_{j+1}$ are indistinguishable even if the adversary doesn't commit to the entire permutation π but only to the values $\pi(i)$ of indices i such that either H_j or H_{j+1} has a pebble on the edge $(v_{\pi(i-1)}, v_{\pi(i)})$. Using our framework, we therefore get a proof of adaptive security where the security loss is $\tau \cdot N^\sigma$ where σ is the maximum number of pebbles used and τ is the number of pebbling moves. In particular, if we use the recursive pebbling strategy described above we only suffer a quasipolynomial security loss $3^{\log N} \cdot N^{\log N + 1}$, as compared with $2N \cdot (N+1)! > (N/e)^N$ for naïve random guessing where the adversary commits to the entire permutation π.

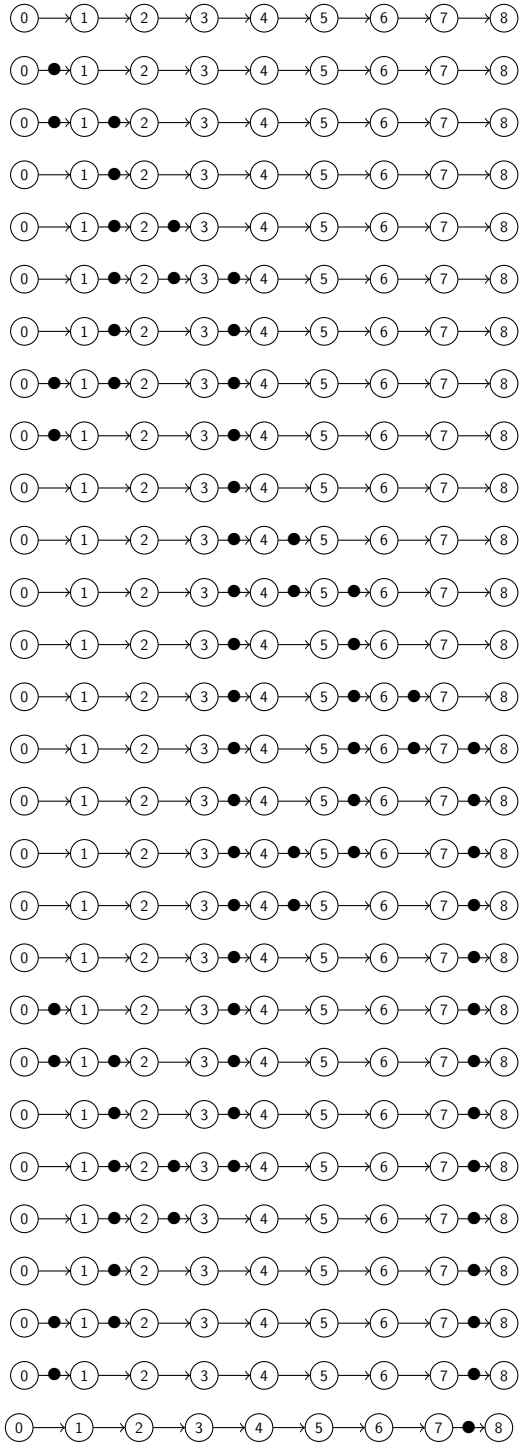
GSD on Low Depth and Other Families of Graphs. The proof outline for GSD on paths is just a very special case of our general result for GSD for various classes of graphs, which we discuss in Section 3.3. If we consider a class of graphs which can be pebbled using τ pebbling configurations, each containing at most σ pebbles, we get a reduction showing that GSD for this class is $\epsilon \cdot \tau \cdot N^\sigma$ secure, assuming the underlying SKE scheme is ϵ-IND-CPA secure.

Unfortunately, this approach will not gain us much for graphs with high in-degree: we can only put a pebble on an edge (v_i, v_j) if all the edges $(*, v_i)$ going into node v_i are pebbled. So if we consider graphs which can have large in-degree δ_{in}, any pebbling strategy must at some point have pebbled all the parents of i, and thus we'll lose at least a factor $N^{\delta_{in}}$ in the reduction. But remember that to apply our Theorem 2, we just need to be able to "compress" the information required to simulate the hybrids. So even if the hybrids correspond to configurations with many pebbles, that is fine as long as we can generate a short hint which will allow to emulate it.

Consider the selective GSD game, where the adversary commits to all of its queries; we can think of this as a DAG, where each edge comes with an index indicating in which query this node was added. Assume the adversary is restricted to choose DAGs of depth D (but no bound on the in-degree). One can show that there exists a pebbling sequence (of length $(2N)^D$), such that in any pebbling configuration, all pebbles lie on a path from a sink to a root (which is of length at most D), and on edges going into this path. Moreover, we can



(a)



(b)

Figure 3.1: "Classical" hybrid argument vs. improved hybrid argument. In both diagrams, the edges that carry a pebble are faked. (a) Illustration of the classical hybrids $\mathsf{H}_0, \ldots, \mathsf{H}_{15}$ for GSD on a path graph with $N = 8$ edges: the number of hybrids is $2N = 16$, and the number of fake edges is at most N. (b) A sequence of hybrids $\tilde{\mathsf{H}}_0, \ldots, \tilde{\mathsf{H}}_{27}$ that use fewer fake edges: even though the number of hybrids is $3^{\log N} + 1 = 28$, the number of fake edges is at most $\log N + 1 = 4$. The argument on the right is identical to the one using nested hybrids in [FJP15], which implicitly uses the edge-pebbling generated by Algorithm 3.4.

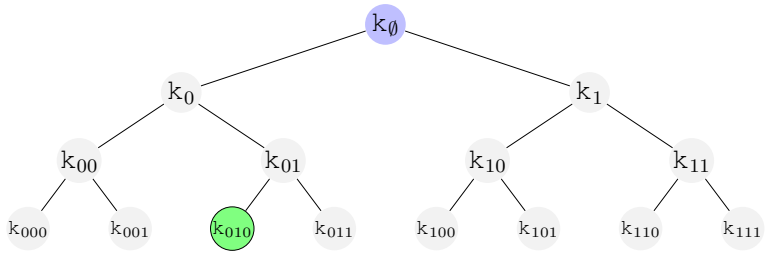


Figure 3.2: Illustration of the GGM PRF. Every left child $k_{x\|0}$ of a node k_x is defined as the first half of $\mathsf{PRG}(k_x)$, the right child $k_{x\|1}$ as the second half. The circled node corresponds to $\mathsf{F}_{\mathsf{GGM}}(k_\emptyset, 010)$.

ensure that in any configuration the following holds: if for a node v_j on this path, there is a pebble on edge (v_i, v_j) with index ι, then all edges of the form $(*, v_j)$ with index $< \iota$ must also have a pebble.

To describe such a configuration, we will output the $\leq D$ nodes on the path, specify for every edge on this path if it is pebbled, and for any node v_j on the path, the number of edges going into v_j that have a pebble (note that there are at most $2^D N^{2D}$ choices for this hint). The hint is sufficient to emulate a hybrid, as for any query $(\texttt{encrypt}, i, j)$ the adversary makes, we will know if the corresponding edge has a pebble or not. This is clear if the edge (i, j) is on the path, as we know this path in full. But also for the other edges that can hold a pebble, where j is on the path but i is not. The reason is that we just have to count which query of the form $(\texttt{encrypt}, *, j)$ this is, as we got a number c telling us that the first c such edges will have a pebble.

Applying Theorem 2, we recover Panjwani's result [Pan07] showing that the GSD game restricted to graphs of depth D only loses a factor $N^{O(D)}$ in the reduction.

3.1.2 Constrained Pseudorandom Functions

Goldreich et al. [GGM84] introduced the notion of a pseudorandom function (PRF). A PRF is an efficiently computable keyed function $\mathsf{F}\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, where $\mathsf{F}(k, \cdot)$, instantiated with a random key $k \leftarrow \mathcal{K}$, cannot be distinguished from a function randomly chosen from the set of all functions $\mathcal{X} \to \mathcal{Y}$ with non-negligible probability. More recently, the notion of constrained pseudorandom functions (cPRF) was introduced as an extension of PRFs, by Boneh and Waters [BW13], Boyle et al. [BGI14] and Kiayias et al. [KPTZ13], independently. Informally, a constrained PRF allows the holder of a master key to derive keys which are constrained to a set, in the sense that such a key can be used to evaluate the PRF on that set, while the outputs on inputs outside of this set remain indistinguishable from random.

Goldreich et al., in addition to formally defining PRFs, gave a construction of a PRF from any length doubling pseudorandom generator (PRG). Their construction is depicted in Figure 3.2. All three of the aforementioned results [BW13, BGI14, KPTZ13] show that this GGM construction already gives a so-called "prefix-constrained" PRF, which is a cPRF where for any $x \in \{0,1\}^*$, one can give out keys which allow to evaluate the PRF on all inputs whose prefix is x. This is a simple but already very interesting class of cPRFs as it can be used to construct a punctured PRF, which in turn is a major tool in constructing various sophisticated primitives based on indistinguishability obfuscation (see, for example, [BW13, SW14, HSW14]).

Prior work. To show that the GGM construction is a prefix-constrained PRF one must show how to transform an adversary that breaks GGM as a prefix-constrained PRF into a distinguisher for the underlying PRG. The proofs in [BW13, BGI14, KPTZ13] only show selective security, where the adversary must initially commit to the output he wants to be challenged on in the security game. There is a loss in tightness by a factor of $2n$, where n is the input length. This can then be turned into a proof against adaptive adversaries via random guessing, losing an additional exponential factor 2^n.

Fuchsbauer et al. [FKPR14] showed that it is possible to achieve adaptive security by losing only a factor of $(3q)^{\log n}$, where q denotes the number of queries made by the adversary – if q is polynomial, the loss is not exponential as before, but just quasi-polynomial. The bound relies on the so-called "nested hybrids" technique. Informally, the idea is to iterate random guessing and hybrid arguments several times. The random guessing is done in a way where one only has to guess some tiny amount of information, which although insufficient to get a full reduction using the hybrid argument, nevertheless reduces the complexity of the task significantly. Every such iteration "cuts" the domain in half, so after logarithmically many iterations the reduction is done. If the number of iterations is small, and the amount of information guessed in each iteration tiny, this can still lead to a reduction with much smaller loss than "single shot" random guessing.

Our results. We cast the result in [FKPR14] in our framework, giving an arguably simpler and more intuitive proof. To this aim, we first describe the GGM construction and sketch its security proof.

Given a PRG: $\{0,1\}^\lambda \to \{0,1\}^{2\lambda}$, the PRF $\mathsf{F}_{\mathsf{GGM}}$: $\{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^\lambda$ is defined recursively as

$$\mathsf{F}_{\mathsf{GGM}}(\mathrm{k}, x) = \mathrm{k}_x \text{ where } \mathrm{k}_\emptyset = \mathrm{k} \text{ and } \mathrm{k}_{x\|0} \| \mathrm{k}_{x\|1} = \mathsf{PRG}(\mathrm{k}_x).$$

The construction is also a prefix-constrained PRF: given a key k_x for any $x \in \{0,1\}^*$, one can evaluate $\mathsf{F}_{\mathsf{GGM}}(\mathrm{k}, x')$ for all x' whose prefix is x.

The security of $\mathsf{F}_{\mathsf{GGM}}$ as a PRF is given in [GGM84]. In particular, they show that if an adversary exists who distinguishes $\mathsf{F}_{\mathsf{GGM}}(\mathrm{k}, \cdot)$ (real experiment) from a uniformly random function (random experiment) with advantage ϵ making q (adaptive) queries, then an adversary of roughly the same time complexity exists who distinguishes $\mathsf{PRG}(U_\lambda)$ from $U_{2\lambda}$ with advantage $\epsilon/(nq)$. Thus if we assume that PRG is ϵ'-secure, then $\mathsf{F}_{\mathsf{GGM}}$ is $\epsilon' nq$-secure against any q-query adversary of the same time complexity. This is one of the earliest applications of the hybrid argument.

The security definition for cPRFs is quite different from that of standard PRFs: the adversary will get to query the cPRF $\mathsf{F}(\mathrm{k}, \cdot)$ in both, the real and random experiment (and can ask for constrained keys, not just regular outputs), and only at the very end the adversary will choose a challenge query x^*, which is then answered with either the correct cPRF output $\mathsf{F}(\mathrm{k}, x^*)$ (in the real experiment) or a random value (in the random experiment). In the selective version of these security experiments, the adversary has to choose the challenge x^* before making any queries. In particular, for the case of prefix-constrained PRFs, the experiment is as follows. The challenger samples $\mathrm{k} \in \{0,1\}^\lambda$ uniformly at random. The adversary A first commits to some $x^* \in \{0,1\}^n$. Then it can make *constrain* queries $x \in \{0,1\}^*$ for any x which is not a prefix of x^*, and receives the constrained key k_x in return. Finally, A gets either $\mathsf{F}_{\mathsf{GGM}}(\mathrm{k}, x^*)$ (in the real game) or a random value, and must guess which is the case.

Selective hybrids. A naïve sequence of selective hybrids, which is of length $2n$, relies just on the knowledge of x^*. For $n = 8$ the corresponding 16 hybrid games are illustrated in Figure 3.1.a. Each path $\mathsf{C}(n)$ corresponds to a hybrid, and it "encodes" how the value of the function F is computed on the challenge input x^* (and this determines how the function is computed on the rest of the inputs too). An edge that does not carry a pebble is computed normally, as defined in $\mathsf{F}_{\mathsf{GGM}}$ – i.e., if the ith edge is not pebbled then $\mathrm{k}_{x^*[1,i-1]\|0}\|\mathrm{k}_{x^*[1,i-1]\|1}$ is set to $\mathsf{PRG}(k_{x[1,i-1]})$, where for $x \in \{0,1\}^n$, $x[1,i]$ denotes its i bit prefix. On the other hand, for an edge with a pebble, we replace the PRG output with a random value – i.e., $\mathrm{k}_{x^*[1,i-1]\|0}\|\mathrm{k}_{x^*[1,i-1]\|1}$ is set to a uniformly random string in $\{0,1\}^{2\lambda}$. It's not hard to see that any distinguisher for two consecutive hybrids can be directly used to break the PRG with the same advantage by embedding the PRG-challenge – which is either $U_{2\lambda}$ or $\mathsf{PRG}(U_\lambda)$ – at the right place. Using standard random guessing we can get adaptive security losing an additional factor 2^n in the distinguishing advantage by initially guessing $x^* \in \{0,1\}^n$.

From selective to adaptive. Before we explain the improved reduction, we take a step back and consider an even more selective game where A must commit, in addition to the challenge query $x_q = x^*$, also to the constrain queries $\{x_1, \ldots, x_{q-1}\}$. We can use the knowledge of $x_1, \ldots, x_{q-1}$ to get a better sequence of hybrids: this requires two tricks. First, as in GSD on a path, instead of using the pebbling strategy in Figure 3.1.a, we switch to the recursive pebbling sequence in Figure 3.1.b. Second, we need a more concise "indexing" for the pebbles: unlike in the proof for GSD, here we can't simply give the positions of the (up to $\log n + 1$) pebbles as hint to simulate the hybrids, as the graph has exponential size; thus even the position of a single pebble would require as many bits to encode as the challenge x^*. Instead, we assume there's an upper bound q on the number of queries made by the adversary. For a pebble on the ith edge, we just give the index of the first constrain query whose i bit prefix coincides with x^*, i.e., the minimum j such that $x_j[1,i] = x^*[1,i]$. This information is sufficient to tell when exactly during the experiment we have to compute a value that corresponds to a pebbled edge.

As there are $3^{\log n}$ hybrids, and each hint comes from a set of size $q^{\log n}$ (i.e., a value $\leq q$ for every pebble), our Theorem 2 implies that $\mathsf{F}_{\mathsf{GGM}}$ is a $\epsilon(3q)^{\log n}$ secure prefix-constrained PRF if PRG is ϵ secure.

3.1.3 Yao's Garbled Circuits

Garbled circuits, introduced by Yao in (oral presentations of) [Yao82, Yao86], can be used to garble a circuit C and an input x in a way that reveals $\mathsf{C}(x)$ but hides everything else. More precisely, a garbling scheme has three procedures; one to garble the circuit C and produce a garbled circuit $\widetilde{\mathsf{C}}$, one to garble the input x and produce a garbled input $\widetilde{x}$, and one that evaluates the garbled circuit $\widetilde{\mathsf{C}}$ on the garbled input $\widetilde{x}$ to get $\mathsf{C}(x)$. Furthermore, to prove security, there must be a simulator that only gets the output of the computation $\mathsf{C}(x)$ and can simulate the garbled circuit $\widetilde{\mathsf{C}}$ and input $\widetilde{x}$, such that no PPT adversary can distinguish them from the real garbling.

Adaptive vs. Selective Security. In the adaptive setting, the adversary A first chooses the circuit C and gets back the garbled circuit $\widetilde{\mathsf{C}}$, then chooses the input x, and gets back garbled input $\widetilde{x}$. The adversary's goal is to decide whether he was interacting with the real garbling scheme or the simulator. In the selective setting, the adversary has to choose the circuit C as well as the input x at the very beginning and only then gets back $\widetilde{\mathsf{C}}, \widetilde{x}$.

Prior Work. The work of Bellare, Hoang and Rogaway [BHR12a] raised the question of whether Yao's construction or indeed any construction of garbled circuits achieves adaptive security. The work of Hemenway et al. [HJO$^+$16] gave the first construction of non-trivial adaptively secure garbled circuits based on one-way functions, by modifying Yao's construction with an added layer of encryption having some special properties. More recently, the work of Jafargholi and Wichs [JW16] gave the first analysis of adaptive security for Yao's (almost) unmodified garbled circuit construction which significantly improves on the parameters of trivial random guessing. See Chapter 6 for a more comprehensive introduction and broader background on garbled circuits and adaptive security.

Here, we present the work of [JW16] as a special case of our general framework. Indeed, the work of [JW16] already implicitly follows our general framework fairly closely and therefore we only give a high level overview of how it fits into it.

Selective Hybrids. We start by outlining the selective security proof for Yao's garbled circuits, following the presentation of [HJO$^+$16, JW16] which is in turn based on the proof of Lindell and Pinkas [LP09]. Essentially the proof proceeds via a series of hybrids which modify one garbled gate at a time from the Real distribution to a Simulated one. However, this cannot be done directly in one step and instead requires going through an intermediate distribution called InputDep (we explain the name later). There are important restrictions on the order in which these steps can be taken:

1. We can switch a gate from Real to InputDep (and vice versa) if it is at the input level or if its predecessor gates are already InputDep.
2. We can switch a gate from InputDep to Simulated (and vice versa) if it is at the output level or if its successor gates are already Simulated.

The simplest strategy to switch all gates from Real to Simulated is to start with the input level and go up one level at a time switching all gates to InputDep. Then start with the output level and go down one level at a time switching all gates to Simulated. This corresponds to the basic proof of selective security of Yao's garbled circuits.

However, the above is not the only possibility. In particular, any strategy for switching all gates from Real to Simulated following rules (1) and (2) corresponds to a sequence of hybrid games for proving selective security. We can identify the above with a *pebbling game* where one can place pebbles on the gates of the circuit. The Real distribution corresponds to not having a pebble and there are two types of pebbles corresponding to the InputDep and Simulated distributions. The goal is to start with no pebbles and finish by placing a Simulated pebble on every gate in the circuit while only performing legal moves according to rules (1) and (2) above. Every pebbling strategy gives rise to a sequence of hybrid games $\mathsf{H}_0, \mathsf{H}_1, \ldots, \mathsf{H}_\tau$ for proving selective security, where the number of hybrids τ corresponds to the number of moves and each hybrid H_i is defined by the configuration of pebbles after i moves.

From Selective to Adaptive. The problem with translating selective security proofs into the adaptive setting lies with the InputDep distribution of a gate. This distribution depends on the input x (hence the name) and, in the adaptive setting, the input x that the adversary will choose is not yet known at the time when the garbled circuit is created. To be more precise, the InputDep distribution of a gate i only depends on the 1-bit value going over the output wire of that gate during the computation $\mathsf{C}(x)$. Moreover, if we take any two fixed hybrid

games $\mathsf{H}_i, \mathsf{H}_{i+1}$ corresponding to two neighbouring pebble configurations (ones which differ by a single move) we can prove indistinguishability even if the adversary does not commit to the entire n-bit input x ahead of time but only commits to the bits going over the output wires of all gates i that are in InputDep mode in either configuration. This means that as long as the pebbling strategy only uses σ pebbles of the InputDep type at any point in time, each pair of hybrids $\mathsf{H}_i, \mathsf{H}_{i+1}$ can be proven indistinguishable in a partially selective setting where the adversary only commits to σ bits of information about their input ahead of time, rather than committing to the entire n-bit input x. Using our framework, this shows that whenever there is a pebbling strategy for the circuit C that requires τ moves and uses at most σ pebbles of the InputDep type, we can translate the selective hybrids into a proof of adaptive security where the security loss is $\tau \cdot 2^{\sigma}$.

It turns out that for any graph of depth D there is a pebbling strategy that uses $O(D)$ pebbles and $\tau = 2^{O(D)}$ moves, meaning that we can prove adaptive security with a $2^{O(D)}$ security loss. This leads to a proof of adaptive security for NC^1 circuits where the reduction has only polynomial security loss, but more generally we can often get a much smaller security loss than the trivial 2^n bound achieved by naïve random guessing.[3]

3.1.4 Related Work

There are several additional results in the literature regarding constructions with tight security analysis that could be placed in our framework. For example, [BKP14, Theorem 3.3] can be viewed as a specific instantiation of our framework with the bit-projection function, that is, hybrid i uses the ith bit of the whole commitment, and therefore the overall degradation in security is 2λ, where λ is the security parameter (the ith bit is guessed in the relevant neighbouring hybrids). Later works on tight security reductions based on the algebraic/adaptive partitioning proof technique [Hof16, Hof17, GHK17] can also be placed under our framework.

After the publication of our work, we learned that Ananth et al. [ACC$^+$16] presented a framework for tight proofs of adaptive security called "small-loss complexity leveraging" and applied it in the context of adaptively-secure RAM delegation of computation. Their framework has similar underlying ideas to ours.

3.2 The Framework

We consider games described via a challenger G which interacts with an adversary A. At the end of the interaction, G outputs a decision bit and we let $\langle \mathsf{G}, \mathsf{A} \rangle$ denote the random variable corresponding to that bit.

Definition 3. We say that two games defined via challengers G_0 and G_1 are (t, ϵ)-indistinguishable, denoted by $\mathsf{G}_0 \approx_{(t,\epsilon)} \mathsf{G}_1$, if for any adversary A running in time at most t:

$$|\Pr[\langle \mathsf{G}_0, \mathsf{A} \rangle = 1] - \Pr[\langle \mathsf{G}_1, \mathsf{A} \rangle = 1]| \leq \epsilon.$$

[3] The presentation in [JW16] follows the above outline fairly closely and the reader can easily match it with our general framework. The one conceptual difference is that we think of all the hybrids H_i as existing in the selective setting where the adversary commits to the entire input but then we analyze indistinguishability of neighbouring hybrids in a partially selective setting. The work of [JW16] thought of the hybrids H_i as already being partially selective, which made it difficult to compare neighbouring hybrids, since the adversary was expected to commit to different information in each one. We view our new framework as being conceptually simpler.

We say that two games are perfectly indistinguishable and write $\mathsf{G}_0 \equiv \mathsf{G}_1$ if they are $(\infty, 0)$-indistinguishable.

Selectivized Games. We define two operations that convert adaptive or partially selective games into further selective games.

Definition 4 (Selectivized Game)**.** Given an (adaptive) game G and some function $\mathsf{w}\colon \{0,1\}^* \to \mathcal{W}$ we define the selectivized game $\mathsf{H} = \mathrm{SEL}_{\mathcal{W}}[\mathsf{G}, \mathsf{w}]$ which works as follows. The adversary A first sends a commitment $w \in \mathcal{W}$ to H. Then H runs the challenger G against A, at the end of which G outputs a bit b'. Let transcript denote all communication exchanged between G and A. If $\mathsf{w}(\text{transcript}) = w$ then H outputs the bit b' and else it outputs 0. See Figure 3.3.(a).

Note that the selectivized game gets a commitment w from the adversary but essentially ignores it during the rest of the game. Only, at the very end of the game, it checks that the commitment matches what actually happened during the game.

Definition 5 (Further Selectivized Game)**.** Assume $\hat{\mathsf{H}}$ is a (partially selective) game which expects to receive some commitment $u \in \mathcal{U}$ from the adversary in the first round. Given functions $\mathsf{w}\colon \{0,1\}^* \to \mathcal{W}$ and $\mathsf{h}\colon \mathcal{W} \to \mathcal{U}$ we define the further selectivized game $\mathsf{H} = \mathrm{SEL}_{\mathcal{U}\to\mathcal{W}}[\hat{\mathsf{H}}, \mathsf{w}, \mathsf{h}]$ as follows. The adversary A first sends a commitment $w \in \mathcal{W}$ to H and H begins running $\hat{\mathsf{H}}$ and passes it $u = \mathsf{h}(w)$. It then continues running the game between $\hat{\mathsf{H}}$ and A at the end of which $\hat{\mathsf{H}}$ outputs a bit b'. Let transcript denote all communication exchanged between $\hat{\mathsf{H}}$ and A. If $\mathsf{w}(\text{transcript}) = w$ then H outputs the bit b' and else it outputs 0. See Figure 3.3.(b).

Note that if $\hat{\mathsf{H}}$ is a (partially selective) game where the adversary sends some commitment u, then in the further selectivized game the adversary might have to commit to more information w. The further selectivized game essentially ignores w and only relies on the partial information $u = \mathsf{h}(w)$ during the course of the game but only at the very end it still checks that the full commitment w matches what actually happened during the game.

Random Guessing. We first present the basic reduction using random guessing.

Lemma 2. *Assume we have two games defined via challengers* G_0 *and* G_1 *respectively. Let* $\mathsf{w}\colon \{0,1\}^* \to \mathcal{W}$ *be an arbitrary function and define the selectivized games* $\mathsf{H}_b = \mathrm{SEL}_{\mathcal{W}}[\mathsf{G}_b, \mathsf{w}]$ *for* $b \in \{0,1\}$*. If* H_0*,* H_1 *are* (t, ϵ)*-indistinguishable then* G_0*,* G_1 *are* $(t - t_{\mathcal{W}}, \epsilon \cdot |\mathcal{W}|)$*-indistinguishable, where* $t_{\mathcal{W}}$ *denotes the time complexity of sampling uniformly at random from* $\mathcal{W}$*.*

Proof. We prove the contrapositive. Assume that there is an adversary A of runtime $t' = t - t_{\mathcal{W}}$ such that

$$|\Pr[\langle \mathsf{A}, \mathsf{G}_0\rangle = 1] - \Pr[\langle \mathsf{A}, \mathsf{G}_1\rangle = 1]| > \epsilon \cdot |\mathcal{W}|.$$

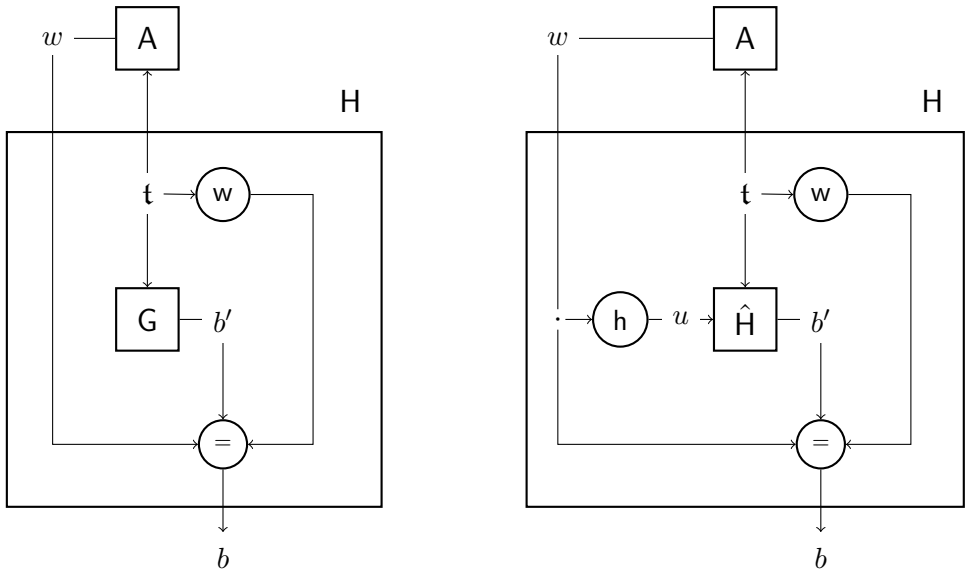
Let A^* be the adversary that first chooses a uniformly random $w \leftarrow \mathcal{W}$ and then runs A. Then for $b \in \{0,1\}$:

$$\Pr[\langle \mathsf{A}^*, \mathsf{H}_b\rangle = 1] = \Pr[\langle \mathsf{A}, \mathsf{G}_b\rangle = 1]/|\mathcal{W}|$$

and therefore

$$|\Pr[\langle \mathsf{A}^*, \mathsf{H}_0\rangle = 1] - \Pr[\langle \mathsf{A}^*, \mathsf{H}_1\rangle = 1]| > \epsilon.$$

Moreover, since A^* runs in time $t' + t_{\mathcal{W}} = t$ this shows that H_0 and H_1 are not (t, ϵ)-indistinguishable. □



(a) (b)

Figure 3.3: *Selectivizing*. (a): $\mathrm{SEL}_{\mathcal{W}}[\mathsf{G}, \mathsf{w}]$, and (b): $\mathrm{SEL}_{\mathcal{U}\to\mathcal{W}}[\hat{\mathsf{H}}, \mathsf{w}, \mathsf{h}]$. The symbol t is short for transcript, the nodes with w and h compute the respective functions, whereas the node with $=$ outputs a bit b as prescribed in the consistency check.

Partially Selective Hybrids.

Consider the following setup. We have two adaptive games G_L and G_R. For some function $\mathsf{w}\colon \{0,1\}^* \to \mathcal{W}$ we define the selectivized games $\mathsf{H}_\mathsf{L} = \mathrm{SEL}_{\mathcal{W}}[\mathsf{G}_\mathsf{L}, \mathsf{w}]$, $\mathsf{H}_\mathsf{R} = \mathrm{SEL}_{\mathcal{W}}[\mathsf{G}_\mathsf{R}, \mathsf{w}]$ where the adversary commits to some information $w \in \mathcal{W}$. Moreover, to show the indistinguishability of $\mathsf{H}_\mathsf{L}, \mathsf{H}_\mathsf{R}$ we have a sequence of τ (selective) hybrid games $\mathsf{H}_\mathsf{L} = \mathsf{H}_0, \mathsf{H}_1, \ldots, \mathsf{H}_\tau = \mathsf{H}_\mathsf{R}$.

If we only assume that neighbouring hybrids $\mathsf{H}_i, \mathsf{H}_{i+1}$ are indistinguishable, then by combining the hybrid argument and random guessing we know that G_L and G_R are indistinguishable at a security loss of $\tau \cdot |\mathcal{W}|$.

Theorem 1. *Assume that for each $i \in \{0, \ldots, \tau - 1\}$, the games $\mathsf{H}_i, \mathsf{H}_{i+1}$ are (t, ϵ)-indistinguishable. Then, G_L and G_R are $(t - t_{\mathcal{W}}, \epsilon \cdot \tau \cdot |\mathcal{W}|)$-indistinguishable, where $t_{\mathcal{W}}$ denotes the time complexity of sampling uniformly at random from $\mathcal{W}$.*

Proof. Follows from Lemma 2 and the hybrid argument (Lemma 1). □

Our goal is to avoid the loss of $|\mathcal{W}|$ in the above theorem. To achieve this, we will assume a stronger condition: not only are neighbouring hybrids $\mathsf{H}_i, \mathsf{H}_{i+1}$ indistinguishable, but they are selectivized versions of less selective games $\hat{\mathsf{H}}_{i,0}, \hat{\mathsf{H}}_{i,1}$ which are already indistinguishable (see Figure 3.4). In particular, we assume that for each pair of neighbouring hybrids $\mathsf{H}_i, \mathsf{H}_{i+1}$ there exist some less selective hybrids $\hat{\mathsf{H}}_{i,0}, \hat{\mathsf{H}}_{i,1}$ where the adversary only commits to much less information $\mathsf{h}_i(w) \in \mathcal{U}$ instead of $w \in \mathcal{W}$. In more detail, for each i there is some function $\mathsf{h}_i\colon \mathcal{W} \to \mathcal{U}$ that lets us interpret H_{i+b} as a selectivized version of $\hat{\mathsf{H}}_{i,b}$ via $\mathsf{H}_{i+b} \equiv \mathrm{SEL}_{\mathcal{U}\to\mathcal{W}}[\hat{\mathsf{H}}_{i,b}, \mathsf{w}, \mathsf{h}_i]$. In that case, the next theorem shows that we only get a security loss proportional to $|\mathcal{U}|$ rather than $|\mathcal{W}|$. Note that different pairs of "less selective hybrids" $\hat{\mathsf{H}}_{i,0}, \hat{\mathsf{H}}_{i,1}$ rely on completely different partial information $\mathsf{h}_i(w)$ about the adversary's choices.

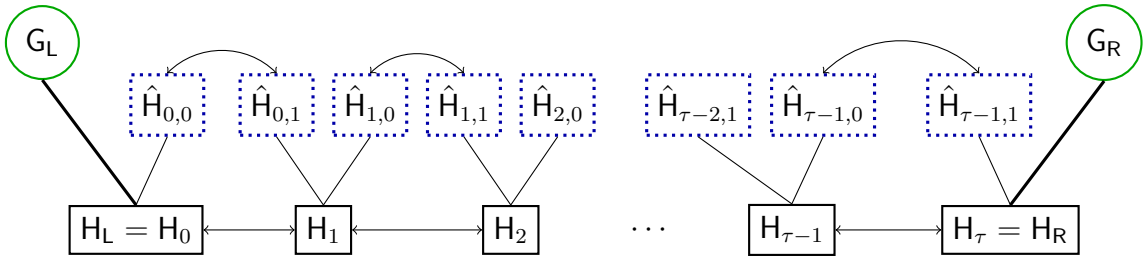


Figure 3.4: A schematic diagram showing the relationship between adaptive, fully selective and partially selective hybrids. The adaptive games $\mathsf{G_L}$ and $\mathsf{G_R}$ are in green (circles); the fully selective games $\mathsf{H}_0, \ldots, \mathsf{H}_\tau$ are in solid black (boxes); and the partially selective games $\hat{\mathsf{H}}_{0,0}, \hat{\mathsf{H}}_{0,1}, \ldots, \hat{\mathsf{H}}_{\tau-1,0}, \hat{\mathsf{H}}_{\tau-1,1}$ are in (dotted) blue boxes. The arrows indicate indistinguishability.

Moreover, the "less selective" hybrid that we associate with each H_i can be different when we compare $\mathsf{H}_{i-1}, \mathsf{H}_i$ (in which case it is $\hat{\mathsf{H}}_{i-1,1}$) and when we compare H_i and H_{i+1} (in which case it is $\hat{\mathsf{H}}_{i,0}$).

Theorem 2 (main)**.** *Let* $\mathsf{G_L}$ *and* $\mathsf{G_R}$ *be two adaptive games. For some function* $\mathsf{w}\colon \{0,1\}^* \to \mathcal{W}$ *we define the selectivized games* $\mathsf{H_L} = \mathrm{SEL}_{\mathcal{W}}[\mathsf{G_L}, \mathsf{w}]$, $\mathsf{H_R} = \mathrm{SEL}_{\mathcal{W}}[\mathsf{G_R}, \mathsf{w}]$. *Let* $\mathsf{H_L} = \mathsf{H}_0, \mathsf{H}_1, \ldots, \mathsf{H}_\tau = \mathsf{H_R}$ *be some sequence of hybrid games.*

Assume that for each $i \in [0, \tau - 1]$*, there exists a function* $\mathsf{h}_i\colon \mathcal{W} \to \mathcal{U}$ *and games* $\hat{\mathsf{H}}_{i,0}, \hat{\mathsf{H}}_{i,1}$ *such that:*

$$\mathsf{H}_i \equiv \mathrm{SEL}_{\mathcal{U}\to\mathcal{W}}[\hat{\mathsf{H}}_{i,0}, \mathsf{w}, \mathsf{h}_i] \quad , \quad \mathsf{H}_{i+1} \equiv \mathrm{SEL}_{\mathcal{U}\to\mathcal{W}}[\hat{\mathsf{H}}_{i,1}, \mathsf{w}, \mathsf{h}_i]. \tag{3.1}$$

Furthermore, assume that $\hat{\mathsf{H}}_{i,0}, \hat{\mathsf{H}}_{i,1}$ *are* (t, ϵ)*-indistinguishable. Then* $\mathsf{G_L}$ *and* $\mathsf{G_R}$ *are* $(t - t_{\mathcal{U}}, \epsilon \cdot \tau \cdot |\mathcal{U}|)$*-indistinguishable, where* $t_{\mathcal{U}}$ *denotes the time complexity of sampling uniformly at random from* $\mathcal{U}$*.*

Proof. Assume that A is an adaptive distinguisher for $\mathsf{G_L}$ and $\mathsf{G_R}$ running in time t' such that

$$|\Pr[\langle \mathsf{A}, \mathsf{G_L}\rangle = 1] - \Pr[\langle \mathsf{A}, \mathsf{G_R}\rangle = 1]| > \epsilon'.$$

Let A^* be a fully selective distinguisher that guesses $w \leftarrow \mathcal{W}$ uniformly at random in the first round and then runs A. By the same argument as in Lemma 2 and Theorem 1 we know that there exists some $i \in [0, \tau - 1]$ such that:

$$|\Pr[\langle \mathsf{A}^*, \mathsf{H}_i\rangle = 1] - \Pr[\langle \mathsf{A}^*, \mathsf{H}_{i+1}\rangle = 1]| \geq \epsilon'/(\tau \cdot |\mathcal{W}|). \tag{3.2}$$

Let A' be a partially selective distinguisher that guesses $u \leftarrow \mathcal{U}$ uniformly at random in the first round and then runs A. We want to relate the probabilities $\Pr[\langle \mathsf{A}^*, \mathsf{H}_{i+b}\rangle = 1]$ and $\Pr[\langle \mathsf{A}', \hat{\mathsf{H}}_{i,b}\rangle = 1]$.

Recall that the game $\langle \mathsf{A}^*, \mathsf{H}_{i+b}\rangle$ consists of A^* selecting a uniformly random value $w \leftarrow \mathcal{W}$ (which we denote by the random variable W) and then we run A against $\hat{\mathsf{H}}_{i,b}(u)$ (denoting the challenger $\hat{\mathsf{H}}_{i,b}$ that gets a commitment u in first round) which results in some transcript and an output bit b^*; if $\mathsf{w}(\text{transcript}) = w$ the final output is b^* else 0.

Similarly, the game $\langle \mathsf{A}', \hat{\mathsf{H}}_{i,b}\rangle$ consists of A' selecting a uniformly random value $u \leftarrow \mathcal{U}$ (which we denote by the random variable U) and then we run A against $\hat{\mathsf{H}}_{i,b}(u)$. Therefore:

$$
\begin{aligned}
&\Pr[\langle \mathsf{A}^*, \mathsf{H}_{i+b}\rangle = 1] \\
&\quad = \sum_{u\in\mathcal{U}} \Pr[\underbrace{\mathsf{h}_i(W) = u}_{\mathsf{I}}] \cdot \Pr[\underbrace{\langle \mathsf{A}, \hat{\mathsf{H}}_{i,b}(u)\rangle = 1}_{\mathsf{II}}] \cdot \Pr[W = \mathsf{w}(\mathsf{transcript})|\mathsf{I},\mathsf{II}] \\
&\quad = \sum_{u\in\mathcal{U}} \frac{|\mathsf{h}_i^{-1}(u)|}{|\mathcal{W}|} \cdot \Pr[\langle \mathsf{A}, \hat{\mathsf{H}}_{i,b}(u)\rangle = 1] \cdot \frac{1}{|\mathsf{h}_i^{-1}(u)|} \\
&\quad = \frac{1}{|\mathcal{W}|} \cdot \sum_{u\in\mathcal{U}} \Pr[\langle \mathsf{A}, \hat{\mathsf{H}}_{i,b}(u)\rangle = 1] \\
&\quad = \frac{|\mathcal{U}|}{|\mathcal{W}|} \cdot \sum_{u\in\mathcal{U}} \Pr[\langle \mathsf{A}, \hat{\mathsf{H}}_{i,b}(u)\rangle = 1] \cdot \Pr[U = u] \\
&\quad = \frac{|\mathcal{U}|}{|\mathcal{W}|} \cdot \Pr[\langle \mathsf{A}', \hat{\mathsf{H}}_{i,b}\rangle = 1].
\end{aligned}
$$

Combining the above with Equation 3.2 we get:

$$|\Pr[\langle \mathsf{A}', \hat{\mathsf{H}}_{i,0}\rangle = 1] - \Pr[\langle \mathsf{A}', \hat{\mathsf{H}}_{i,1}\rangle = 1]| \geq \epsilon'/(\tau \cdot |\mathcal{U}|).$$

Since by assumption $\hat{\mathsf{H}}_{i,0}, \hat{\mathsf{H}}_{i,1}$ are (t,ϵ)-indistinguishable and A' is running in time $t' + t_{\mathcal{U}}$ this shows that when $t' = t - t_{\mathcal{U}}$ then $\epsilon' \leq \epsilon \cdot \tau \cdot |\mathcal{U}|$ which proves the theorem.

□

3.2.1 Example: GSD on a Path

As an example, we consider the problem of generalized selective decryption (GSD) on a path graph with N edges, where N is a power of two.

Let $(\mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme with (probabilistic) $\mathsf{Enc}\colon \mathcal{K}\times\mathcal{M}\to\mathcal{C}$ and $\mathsf{Dec}\colon \mathcal{K}\times\mathcal{C}\to\mathcal{M}$. We assume $\mathcal{K}\subseteq\mathcal{M}$ so that we can encrypt keys, and that the encryption scheme is (t,ϵ)-indistinguishable under chosen-plaintext attack.[4] In the game, the challenger – either G_L or G_R – picks $N+1$ random keys $\{\mathtt{k}_0,\ldots,\mathtt{k}_N\}\in\mathcal{K}$, and the adversary A is then allowed to make two types of queries:

- Encryption queries, $(\mathtt{encrypt}, i, j)$: it receives back $\mathsf{Enc}_{\mathtt{k}_i}(\mathtt{k}_j)$.
- Challenge query, $(\mathtt{challenge}, i^*)$: here the answer differs between G_L and G_R, with G_L answering with $\mathtt{k}_{i^*}$ (real key) and G_R answering with $\mathtt{r}\leftarrow\mathcal{K}$ (random, "fake" key).

A cannot ask arbitrary queries: it is restricted to encryption queries that form a path graph with the challenge query as the sink. That is, a valid attacker A is allowed exactly N encryption queries $(\mathtt{encrypt}, i_\iota, j_\iota)$, for $\iota\in[1,N]$, and a single $(\mathtt{challenge}, i^*)$ query such that the directed graph $G=(\mathcal{V},\mathcal{E})$ with $\mathcal{V}=\{v_0,\ldots,v_N\}$ and $\mathcal{E}=\{(v_{i_1},v_{j_1}),\ldots,(v_{i_N},v_{j_N})\}$ forms a path with sink v_{i^*}.

[4] To be precise, we only need the encryption scheme to be secure in a weaker model where encryptions of two random messages $\mathtt{m}_0,\mathtt{m}_1\in\mathcal{K}$ under a random key $\mathtt{k}\in\mathcal{K}$ are (t,ϵ)-indistinguishable, with the adversary having access to ciphertexts on random messages from $\mathcal{K}$.

Fully selective hybrids. Let's look at a naïve sequence of intermediate hybrids $\{\mathsf{H}_0, \ldots, \mathsf{H}_{2N-1}\}$. The fully selective challenger H_I receives as commitment the exact permutation π that A will query – i.e, $v_{\pi(i)}$ is the ith vertex on the path. Therefore, $\mathcal{W} = \mathbb{S}_{N+1}$ (the symmetry group over $[0, N]$) and w is the function that outputs the observed permutation from the transcript. Next, H_I samples $2(N+1)$ keys $\{\mathtt{k}_0, \ldots, \mathtt{k}_N\}, \{\mathtt{r}_0, \ldots, \mathtt{r}_N\}$, and when A makes a query $(\mathtt{encrypt}, \pi(i), \pi(i+1))$, it returns
for $0 \leq I \leq N$:

$$\begin{array}{lll} \mathsf{Enc}_{\mathtt{k}_{\pi(i)}}(\mathtt{r}_{\pi(i+1)}) & \text{if } (0 \leq i \leq I) & \text{(Fake edge)} \\ \mathsf{Enc}_{\mathtt{k}_{\pi(i)}}(\mathtt{k}_{\pi(i+1)}) & \text{otherwise,} & \text{(Real edge)} \end{array}$$

for $N < I \leq 2N-1$:

$$\begin{array}{lll} \mathsf{Enc}_{\mathtt{k}_{\pi(i)}}(\mathtt{r}_{\pi(i+1)}) & \text{if } (0 \leq i \leq 2N-1-I) \vee (i = N-1) & \text{(Fake edge)} \\ \mathsf{Enc}_{\mathtt{k}_{\pi(i)}}(\mathtt{k}_{\pi(i+1)}) & \text{otherwise.} & \text{(Real edge)} \end{array} \tag{3.3}$$

Thus, in the sequence $\{\mathsf{H}_0, \ldots, \mathsf{H}_{2N-1}\}$, edges are "faked" sequentially down the path, and then "restored", except for the last edge, in the reverse order up the path – see Figure 3.1.a. By definition, $\mathsf{H}_0 = \mathsf{G}_\mathsf{L}$ and $\mathsf{H}_{2N-1} = \mathsf{G}_\mathsf{R}$. Moreover, H_I and H_{I+1} can be shown (t, ϵ)-indistinguishable: when A queries for $(\mathtt{encrypt}, \pi(I), \pi(I+1))$, the reduction R_I returns the challenge ciphertext

$$\begin{array}{lll} \mathsf{C}(\cdot, \mathtt{k}_{\pi(I+1)}, \mathtt{r}_{\pi(I+1)}) & \text{if } (I \leq n) & \text{(Real to fake)} \\ \mathsf{C}(\cdot, \mathtt{r}_{\pi(I+1)}, \mathtt{k}_{\pi(I+1)}) & \text{otherwise.} & \text{(Fake to real)} \end{array} \tag{3.4}$$

For the rest of the queries, R_I works as prescribed in eq.3.3.[5] It is easy to see that R_I simulates H_I when the ciphertext corresponds to the first message, and H_{I+1} otherwise. By Theorem 1, $(t - N \cdot t_{\mathsf{Enc}}, \epsilon(2N+1)(N+1)!)$-indistinguishability of G_L and G_R follows, where t_{Enc} is the time complexity of the Enc algorithm and the $(N+1)!$ factor is the size of the set $\mathcal{W} = \mathbb{S}_{N+1}$.

Partially selective hybrids. In order to simulate according to the strategy just described, it suffices for the hybrid (as well as the reduction) to guess the edges that are faked – however, this number can be up to N (e.g. in the middle hybrids) and, therefore, the simulator guesses the whole path anyway. Intuitively, this is where the overall looseness of the bound stems from. Now, consider the alternative sequence of hybrids $\tilde{\mathsf{H}}_0, \ldots, \tilde{\mathsf{H}}_{27}$ given in Figure 3.1.b: the edges in this sequence are faked and restored, one at a time, in a recursive manner to ensure that at most four edges end up fake per hybrid. In particular, the new hybrid $\tilde{\mathsf{H}}_I$, fakes all the edges that belong to a set $\mathcal{P}_I \subseteq \mathcal{E}$. That is, when A makes a query $(\mathtt{encrypt}, i, j)$ – instead of following Eq.3.3, – $\tilde{\mathsf{H}}_I$ returns

$$\begin{array}{lll} \mathsf{Enc}_{\mathtt{k}_i}(\mathtt{r}_j) & \text{if } ((v_i, v_j) \in \mathcal{P}_I) & \text{(Fake edge)} \\ \mathsf{Enc}_{\mathtt{k}_i}(\mathtt{k}_j) & \text{otherwise.} & \text{(Real edge)} \end{array} \tag{3.5}$$

This strategy can be extended to arbitrary N, and there exists such a sequence of sets $\mathcal{P}_0, \ldots, \mathcal{P}_{3^{\log N}}$ where the sets are of size at most $\log N + 1$.[6]

Next, we show that the above simulation strategy satisfies the requirements for applying Theorem 2. Firstly, as shown in Algorithm 3.1, the strategy is partially selective – i.e., $\tilde{\mathsf{H}}_{I+b} = \mathrm{SEL}_{\mathcal{U}\to\mathcal{W}}[\hat{\mathsf{H}}_{I,b}, \mathsf{w}, \mathsf{h}_I]$, where, for $I \in [0, \tau = 3^{\log N}]$, the function $\mathsf{h}_I : S_{N+1} \to \mathcal{E}^{\log N+1}$ computes $\mathcal{P}_I$.

[5] Even though R_I does not know the key $\mathtt{k}_{\pi(I)}$, the query $(\mathtt{encrypt}, v_{\pi(I-1)}, v_{\pi(I)})$ does not cause a problem as its response is $\mathsf{Enc}_{\mathtt{k}_{\pi(I-1)}}(\mathtt{r}_{\pi(I)})$.

[6] Later, in Section 3.3.2, we see that the sequence $\mathcal{P}_0, \ldots, \mathcal{P}_{3^{\log N}}$ corresponds to an "edge-pebbling" of the path graph.

Algorithm 3.1: Hybrid $\tilde{\mathsf{H}}^{\mathsf{A}}_{I+b}$, where $\tilde{\mathsf{H}}_{I+b} = \mathrm{SEL}_{\mathcal{U}\to\mathcal{W}}[\hat{\mathsf{H}}_{I,b}, \mathsf{w}, \mathsf{h}_I]$

$\tilde{\mathsf{H}}^{\mathsf{A}}_{I+b}$
Obtain $\pi \in \mathbb{S}_{N+1}$ from A
Compute $\mathcal{P} := \{\mathcal{P}_0, \ldots, \mathcal{P}_\tau\}$
Run $\hat{\mathsf{H}}^{\mathsf{A}}_{I,b}((\mathcal{P}_I, \mathcal{P}_{I+1}))$
if $\mathsf{w}(\text{transcript}) = \pi$ **then**
  return $\hat{\mathsf{H}}_{I,b}$'s output
else
  return 0

$\hat{\mathsf{H}}^{\mathsf{A}}_{I,b}((\mathcal{P}_I, \mathcal{P}_{I+1}))$
Choose $2N$ keys $\{\mathtt{r}_1, \ldots, \mathtt{r}_N\}, \{\mathtt{k}_1, \ldots, \mathtt{k}_N\} \leftarrow \mathcal{K}$
Whenever A queries $(\mathtt{encrypt}, i, j)$:
if $(v_i, v_j) \in \mathcal{P}_{I+b}$ **then**
  return $\mathsf{Enc}_{\mathtt{k}_i}(\mathtt{r}_j)$
else
  return $\mathsf{Enc}_{\mathtt{k}_i}(\mathtt{k}_j)$
return A's output

Secondly, as the simulations in $\hat{\mathsf{H}}_{I,0}$ and $\hat{\mathsf{H}}_{I,1}$ differ by exactly one edge – which is real in one and fake in the other – they can be shown to be (t, ϵ)-indistinguishable. To be precise, if $(v_{i^*}, v_{j^*}) := \mathcal{P}_I \triangle \mathcal{P}_{I+1}$, where $\triangle$ denotes the symmetric difference, when A queries for $(\mathtt{encrypt}, i^*, j^*)$, the reduction $\tilde{\mathsf{R}}_I$ returns

$$\begin{array}{lll} \mathsf{C}(\cdot, \mathtt{k}_{j^*}, r_{j^*}) & \text{if } (\mathcal{P}_I \subset \mathcal{P}_{I+1}) & \text{(Real to fake)} \\ \mathsf{C}(\cdot, r_{j^*}, \mathtt{k}_{j^*}) & \text{otherwise.} & \text{(Fake to real)} \end{array} \tag{3.6}$$

with the rest of the queries answered as in Eq.3.5.

Although the number of hybrids is greater than in the previous sequence, the number of fake edges in any hybrid is at most $\log N + 1$. Thus, the reduction can work with less information than earlier. By Theorem 2, $(t - N \cdot t_{\mathsf{Enc}} - t_{\mathcal{P}}, \epsilon \cdot 3^{\log N} \cdot N^{2(\log N+1)})$-indistinguishability of G_L and G_R follows, where $t_{\mathcal{P}}$ is the run time of the algorithm that generates the set $\mathcal{P} = \{\mathcal{P}_0, \ldots, \mathcal{P}_\tau\}$, and the $N^{2(\log N+1)}$ factor results from the fact that the compressed set $\mathcal{U} = \mathcal{E}^{\log N+1}$. Thus, the bound is improved considerably from exponential to quasi-polynomial. A more formal treatment is given in the following section.

3.3 Application I: Generalized Selective Decryption

The generalized selective decryption (GSD) game was introduced in [Pan07] in order to capture the hardness of proving adaptive security of cryptographic protocols. [Pan07] then showed how GSD can be used to show adaptive security of multicast encryption protocols such as a variant of the logical key hierarchy [WGL00].

3.3.1 Formal Definitions

We generalize the definition of GSD given in Section 3.2.1. Let $(\mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme with $\mathsf{Enc}\colon \mathcal{K} \times \mathcal{M} \to \mathcal{C}$, $\mathsf{Dec}\colon \mathcal{K} \times \mathcal{C} \to \mathcal{M}$ and we assume $\mathcal{K} \subseteq \mathcal{M}$ (so we can encrypt keys). We assume that $(\mathsf{Enc}, \mathsf{Dec})$ is correct and (t, ϵ)-IND-CPA secure – see Definition 1.

Definition 6 (Adaptive GSD [Pan07])**.** The game is played between a challenger G (which is either G_L or G_R) and an adversary A using $(\mathsf{Enc}, \mathsf{Dec})$. G picks N keys $\mathtt{k}_1, \ldots, \mathtt{k}_N \leftarrow \mathcal{K}$ uniformly at random, and initialises a graph $G := (\{v_1, \ldots, v_N\}, \emptyset)$; it also initialises a set $\mathcal{V}^{\mathsf{cor}} = \emptyset$. A can make three types of queries:

- Encryption queries, $(\mathtt{encrypt}, i, j)$: G returns $\mathsf{Enc}_{\mathtt{k}_i}(\mathtt{k}_j)$, and adds (v_i, v_j) to $\mathcal{E}$. Here, A is restricted to queries such that $\mathcal{E}$ does not contain any cycles.[7]
- Corruption queries, $(\mathtt{corrupt}, i)$: G returns $\mathtt{k}_i$, and adds v_i to $\mathcal{V}^{\mathsf{cor}}$.
- One challenge query $(\mathtt{challenge}, i)$: Here the answer differs between G_L and G_R: G_L answers with $\mathtt{k}_i$ (real key), whereas G_R answers with $\mathtt{r} \leftarrow \mathcal{K}$ (fake key) sampled uniformly at random – for the task to be non-trivial, v_i must be a sink and it *must not* be reachable from any vertex in $\mathcal{V}^{\mathsf{cor}}$.[8]

We consider an order "$<$"on $\mathcal{E}$, which is the "temporal" order induced by the game: if A queried $(\mathtt{encrypt}, i, j)$ before $(\mathtt{encrypt}, i', j')$ in the game, then (v_i, v_j) was added to the set $\mathcal{E}$ before $(v_{i'}, v_{j'})$, and hence $(v_i, v_j) < (u_{i'}, v_{j'})$.[9] The graph G, along with the order $<$, is called the *key-graph*. In the fully selectivized version of Definition 6, A must commit to the key-graph beforehand – i.e., the selective challenger $\mathsf{H}_\mathsf{L} = \mathrm{SEL}_{\mathcal{G}}[\mathsf{G}_\mathsf{L}, \mathsf{w}]$ (resp., $\mathsf{H}_\mathsf{R} = \mathrm{SEL}_{\mathcal{G}}[\mathsf{G}_\mathsf{R}, \mathsf{w}]$), where $\mathcal{G} = \mathcal{G}(N)$ is the set of all key-graphs of N vertices (i.e., the set of all graphs $\mathcal{G}(N)$ along with the set of all possible edge orderings on them), and w is the function that extracts the key-graph from the transcript.

Definition 7. An encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ is called (t, ϵ)-adaptive (resp., selective) GSD-secure if G_L and G_R (resp., H_L and H_R) are (t, ϵ)-indistinguishable.

Existing Results. Let t_{Enc} denote the run time of the algorithm Enc. [Pan07] shows that if the key-graph is of depth $D = D(N)$, then an (t, ϵ)-indistinguishable encryption scheme is also $(t - (N \cdot t_{\mathcal{K}} + N^2 \cdot t_{\mathsf{Enc}}), O(\epsilon \cdot (2N)^{D+1}))$-adaptive GSD-secure. As a corollary, for perfect binary trees the loss in tightness is only $O(N^{\log N + 2})$. In [FJP15], it is shown that if the key-graph is a tree, then the encryption scheme is $(t - (N \cdot t_{\mathcal{K}} + N^2 \cdot t_{\mathsf{Enc}}), O(\epsilon \cdot N^{O(\log N)}))$-adaptive GSD-secure – the result is established using the nested hybrids technique [FKPR14]. We recapture the result[10] in [Pan07], as well as the result for paths in [FJP15] in our framework.

[7] Without this restriction, for security we would have to assume the encryption scheme is *circular secure* – a strictly stronger security notion than IND-CPA security [MO14].

[8] As noted in [Pan07], through a standard hybrid argument, ϵ-security in the above model implies $(\epsilon \cdot m)$-security in a (stronger) model where $m = m(\lambda)$ challenges are allowed.

[9] An order can be maintained even when there are parallel queries (viz., the order within the parallel query).

[10] In fact, we prove a slightly weaker result, however, the proof in our framework is *significantly* simpler.

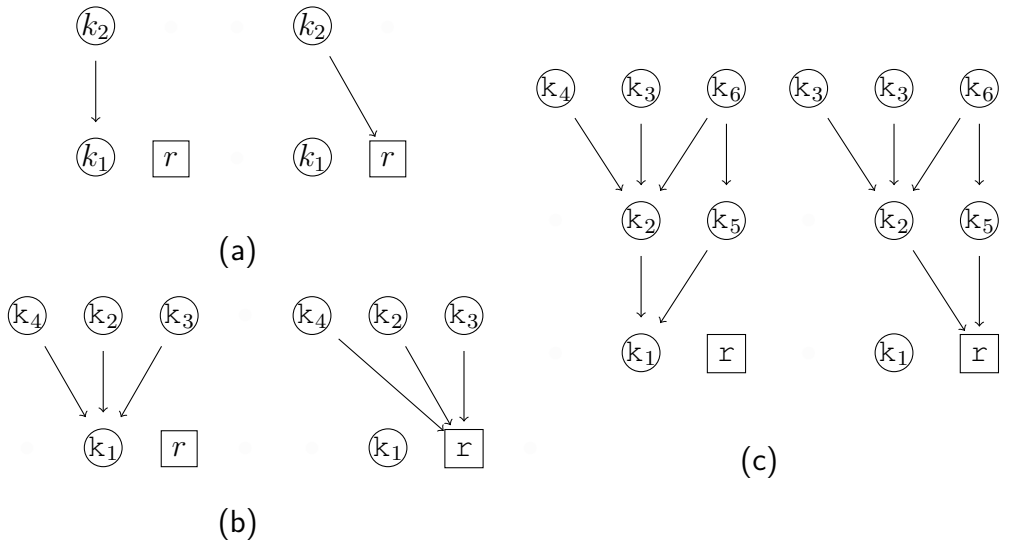


Figure 3.5: Examples of key-graphs for the extreme games H_L and H_R. (a): G_1, (b): G_2 and (c): G_3. In the key-graphs given on the left (the left key-graph), all the edges are real, whereas in those given on the right (the right key-graph), only the edges that are incident on the challenge vertex (v_1) are fake.

Overview.

We prove selective security first (c.f., Section 3.3.2) and then apply Theorem 2 to establish adaptive security (c.f., Theorem 3). The main idea behind proving selective security is to associate a hybrid experiment to a pebbling configuration of the underlying key-graph (c.f., Lemma 3).

3.3.2 Hybrids and Pebbling Configurations

Recall that our goal is to show that the indistinguishability of the encryption scheme implies indistinguishability of the fully selectivized games H_L and H_R. To this aim, we first construct a set of $\tau - 1$ intermediate (fully selective) hybrids $\mathsf{H}_\mathsf{L} = \mathsf{H}_0, \mathsf{H}_1, \ldots, \mathsf{H}_\tau = \mathsf{H}_\mathsf{R}$ by associating each experiment with a pebbling configuration. Then we further selectivize these games by showing that there exists $\hat{\mathsf{H}}_{i,b}$, $b \in \{0,1\}$, for each H_i, $i \in [0,\tau]$. For this, we rely on the pebbling sequence to have certain desirable properties.

Fully Selective Hybrids.

Let's focus on the first part. For ease of exposition, let's restrict the adversary to commit key-graphs having only one sink which, by definition, is also the challenge vertex in the GSD game. Now, consider the structure of key-graphs for H_L and H_R – let's call them, respectively, the left and the right key-graph. In the left key-graph all the edges are real, whereas in the right key-graph only the edges that are incident on the sink are fake – some examples are given in Figure 3.5:

Consider the simplest case of G_1: the left and right key-graphs correspond, exactly, to the left and right games in IND-CPA. To be precise, they correspond to the IND-CPA game where k_2 is the challenge key, and the adversary challenges on the messages $(\mathrm{k}_1, \mathrm{r})$: if the challenger responds with a ciphertext that corresponds to k_1, then we end up with the left key-graph for

G_1, and otherwise we end up with the right key-graph. Thus, in the case of G_2, changing an edge from real to fake is indistinguishable to a GSD adversary. For G_2, the edges incident on v_1 have to be faked iteratively: first (v_4, v_1), then (v_2, v_1) and finally (v_3, v_1), and we end up in the right key-graph. Thus, we get a sequence of hybrids $\mathsf{H}_\mathsf{L} = \mathsf{H}_0, \ldots, \mathsf{H}_3 = \mathsf{H}_\mathsf{R}$, one corresponding to each act of faking. Each of these moves can be shown indistinguishable by reducing to the case of G_1.

Next consider the slightly more involved graph G_3: unlike in the cases of G_1 and G_2, some of the edges in G_3 *cannot* be faked straight away. For example, consider the left key-graph and the key-graph that has (v_2, v_1) faked. An attempt to show that these two are indistinguishable would fail: such a reduction, which must set $\mathtt{k}_2$ as the challenge key, would not be able to answer, for example, to the query $(\mathtt{encrypt}, 4, 2)$. However, if we first fake all the edges that are incident on v_2 (iteratively, as in the case of G_2), then the query $(\mathtt{encrypt}, 4, 2)$ does not pose a problem (as it is responded with $\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{r})$ for some $\mathtt{r} \leftarrow \mathcal{K}$). Thus, before faking an edge (u, v) we must ensure that all the incoming edges to u are faked. Such a sequence is given in Figure 3.6.

To summarize, our goal is to start with the left key-graph and end up with the right key-graph by faking edges or restoring faked edges, *one* at a time keeping in mind that before faking or restoring an edge (u, v) all the edges coming in to u must be already fake. This can be abstracted out as an edge-pebbling game: faking (resp., restoring) an edge is equivalent to placing (resp., removing) a pebble on the edge, and the goal of the pebbling game is to pebble all the incoming edges of the sink. A more formal definition follows.

Reversible edge-pebbling. The classical reversible black pebbling game on DAGs was introduced in [Ben89] to model reversible computation. A pebble in the reversible *edge*-pebbling game – unlike in the classical case – is placed on the edges. Thus, a pebbling configuration is a subset of the edges. A *vertex* is deemed pebbled if all its incoming edges are pebbled – i.e., $v \in \mathcal{V}$ is pebbled in a configuration $\mathcal{P}_i$ if $\mathsf{in}(v) \subseteq \mathcal{P}_i$.

Definition 8. A *reversible edge-pebbling* of a DAG $G = (\mathcal{V}, \mathcal{E})$ with source nodes $\mathcal{S}$ and sink nodes $\mathcal{T}$ is a sequence $\boldsymbol{\mathcal{P}} := (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ with each configuration $\mathcal{P}_i \subseteq \mathcal{E}$ and, in particular, $\mathcal{P}_0 = \emptyset$ and $\mathcal{P}_\tau = \cup_{v \in \mathcal{T}'} \mathsf{in}(v)$ for some $\emptyset \neq \mathcal{T}' \subseteq \mathcal{T}$. In a move, a pebble can be placed on or removed from an edge (u, v) iff the vertex u is pebbled – edges going out from $\mathcal{S}$ can be pebbled or unpebbled in any move. Thus, $\boldsymbol{\mathcal{P}}$ is a valid pebbling sequence iff

$$\forall i \in [1, \tau] : \exists! \, (u, v) \in \mathcal{P}_{i-1} \triangle \mathcal{P}_i \text{ and } \mathsf{in}(u) \subseteq \mathcal{P}_{i-1}.$$

Thus, for a target set $\mathcal{T}' \subseteq \mathcal{T}$, starting from a completely unpebbled graph, the aim is to achieve a pebbling configuration in which only the vertices $\mathcal{T}'$ are pebbled, and all other edges are unpebbled. The space complexity of an edge pebbling $\boldsymbol{\mathcal{P}}$ is defined as $\sigma_{\boldsymbol{\mathcal{P}}}(G) := \max_{i \in [0,\tau]} |\mathcal{P}_i|$, and the space complexity of edge-pebbling a DAG is $\sigma(G) := \min_{\boldsymbol{\mathcal{P}}} \sigma_{\boldsymbol{\mathcal{P}}}(G)$. Similarly, for an edge-pebbling $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ of a graph G, the number of moves (τ, i.e.,) is denoted by $\tau_{\boldsymbol{\mathcal{P}}}(G)$.

Definition 9 (Ordered edge-pebbling)**.** An edge-pebbling $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ of a graph $G = (\mathcal{V}, \mathcal{E})$ that has ordered $\mathcal{E}$ is said to be *ordered* if for each configuration $\mathcal{P}_i$ the following holds:

$$(u, v) \in \mathcal{P}_i \implies \forall \, ((u', v) < (u, v)) : (u', v) \in \mathcal{P}_i.$$

In other words, if an edge (u, v) carries a pebble in $\mathcal{P}_i$ then all the edges that are incident on v and *precede* (u, v) must also carry pebbles.

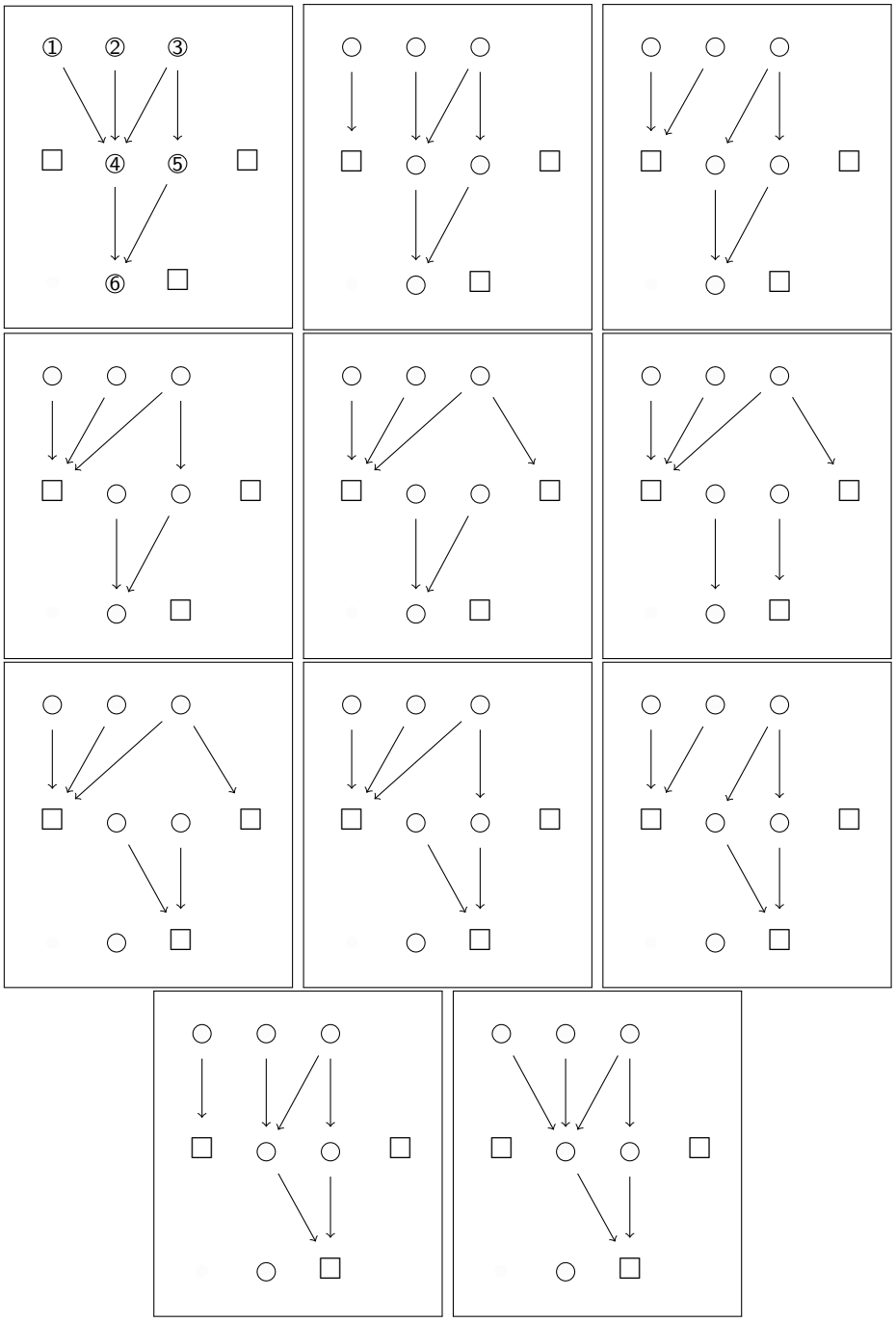


Figure 3.6: A sequence of hybrids for G_3. The square boxes indicate fake keys and, hence, the edges incident on these boxes are the faked edges.

For an ordered edge-pebbling, we are interested in a coarser measure of space complexity called the *lateral* space complexity, which takes into account the fact that an ordered pebbling is "compressible": in a configuration $\mathcal{P}_i$, if the pebbles that come in to a vertex v are $(u_1, v), \ldots, (u_p, v)$ (in that order), then it suffices just to store v and the number of incident pebbled edges, i.e., p. Note that for a graph with bounded indegree δ_{in}, the integer p is bounded by δ_{in}. We formalize this intuition in the following definition.

Definition 10. Let $G = (\mathcal{V}, \mathcal{E})$ be a DAG, $v \in \mathcal{V}$, and $\mathcal{P}_i$ an ordered pebbling configuration

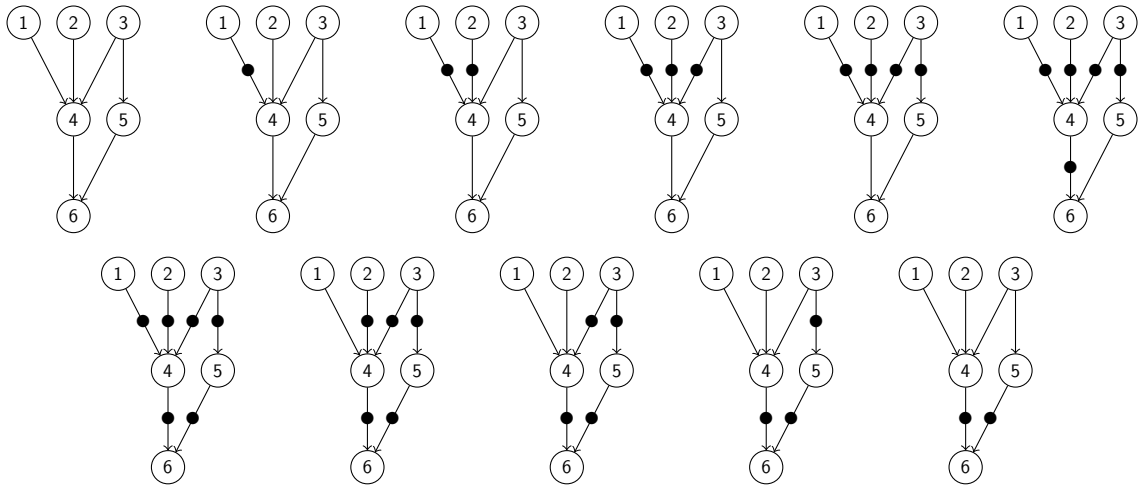


Figure 3.7: An example of an edge-pebbling sequence – it corresponds to a sequence of hybrids for the graph G_3. If an edge carries a pebble, then that edge is faked during the simulation. Thus, the first configuration is the real game, whereas the last configuration – with all the incoming edges to the challenge faked – is the random game.

on G. Then $\mathsf{index}(v, \mathcal{P}_i)$ denotes the number of incoming edges to v that are pebbled – i.e.,

$$\mathsf{index}(v, \mathcal{P}_i) := |\{(u, v') \in \mathcal{P}_i : v' = v\}| .$$

Let the set $\mathcal{Q}_i$ be defined as

$$\mathcal{Q}_i := \{(v, \mathsf{index}(v, \mathcal{P}_i)) : v \in \mathcal{V}, \mathsf{index}(v, \mathcal{P}_i) > 0\}.$$

The *lateral space-complexity* σ' of the edge pebbling is defined as the maximum size of these sets – i.e., $\sigma'_{\boldsymbol{\mathcal{P}}}(G) := \max_{i \in [0,\tau]} |\mathcal{Q}_i|$. The lateral space complexity of edge-pebbling a DAG is $\sigma'(G) := \min_{\mathcal{P}} \sigma'_{\boldsymbol{\mathcal{P}}}(G)$.[11]

The edge-pebbling equivalent to the sequence of hybrids given in Figure 3.6 is given in Figure 3.7. However, it is *not* ordered. An alternative edge-pebbling, which is ordered, is given in Figure 3.8. Next, we formally show that an ordered edge-pebbling implies a sequence of fully selective hybrids.

Lemma 3. *For $G \in \mathcal{G}$, let $(\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ be an ordered edge-pebbling that is generated by a deterministic edge-pebbling algorithm* P*. Then the sequence of hybrids $\mathsf{H}_{\mathsf{P},I}$, $I \in [0, \tau]$ and with $\mathsf{H}_{\mathsf{P},I}$ defined in Algorithm 3.2, constitutes a valid sequence of fully selective hybrids. Furthermore, if t_{P} denotes the time complexity of* P*, then an encryption scheme* (Enc, Dec) *that is (t, ϵ)-secure under IND-CPA, is $(t - O(t_{\mathsf{P}} + N \cdot t_{\mathcal{K}} + N^2 \cdot t_{\mathsf{Enc}}), \epsilon \cdot \tau)$-selective GSD-secure.*

As a corollary to Lemma 3 and Theorem 1, an encryption scheme (Enc, Dec) that is (t, ϵ)-secure under IND-CPA, is $(t - O(t_{\mathsf{P}} + N \cdot t_{\mathcal{K}} + N^2 \cdot t_{\mathsf{Enc}} + t_{\mathcal{W}}), \epsilon \cdot \tau \cdot \exp(N))$-adaptive GSD-secure.

Proof of lemma. Fix a graph G and consider the pebbling sequence $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau) \leftarrow \mathsf{P}(G)$. Each configuration $\mathcal{P}_I$ yields a challenger $\mathsf{H}_{I,\mathsf{P}}$ described in Algorithm 3.2. By the

[11] Note that $\sigma'(G) \leq \sigma(G)$, with equality holding, for example, for graphs with in-degree one – also, $\sigma(G) \leq \delta_{in} \cdot \sigma'(G)$, where δ_{in} is the in-degree of the DAG.

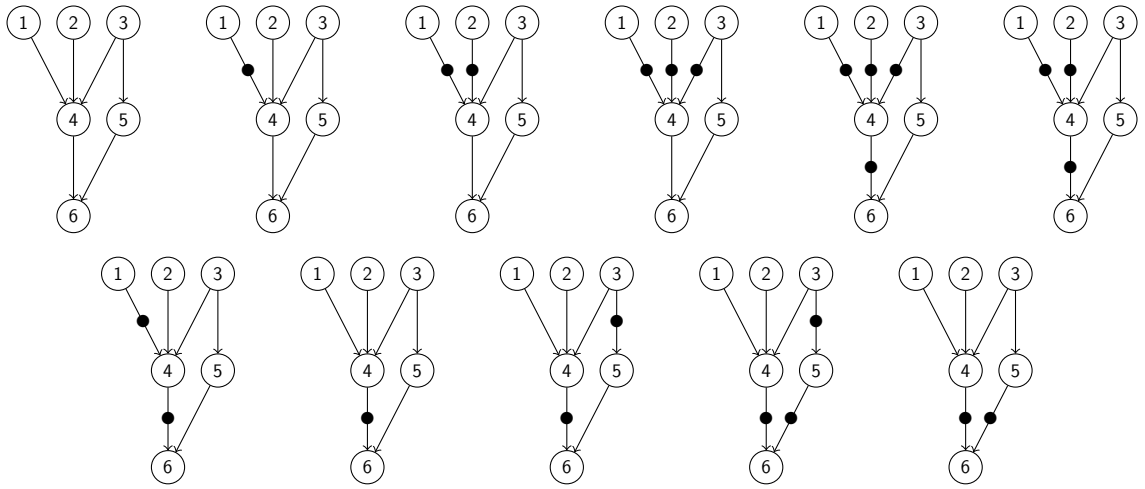


Figure 3.8: An example of edge-pebbling of G_3 that uses "fewer" pebbles than in Figure 3.7. Although the space complexity of the above DAG is four, its lateral space complexity is just two. Note that the edge-pebbling is ordered.

Algorithm 3.2: Template for generating fully selective hybrids.

$\mathsf{H}^{\mathsf{A}}_{\mathsf{P},I}$

Obtain the key-graph $G \in \mathcal{G}$ from A
Compute $\mathcal{P}_0, \ldots, \mathcal{P}_\tau \leftarrow \mathsf{P}(G)$ `// Generate the ordered pebbling`
Initialise $c_1, \ldots, c_N := 0$ `// Counters for edges incident on each vertex`
Compute $\mathcal{Q}_I$ `// Contains set of "pebbled" vertices and their indices`
Choose $2N$ keys $\mathtt{r}_1, \ldots, \mathtt{r}_N, \mathtt{k}_1, \ldots, \mathtt{k}_N \leftarrow \mathcal{K}$
Whenever A makes a query $(\mathtt{encrypt}, v_i, v_j)$:
if $(v_j \in \mathcal{Q}_I)$ **then** `// Carries pebble?`
  if $c_j \leq \mathit{index}_j$ **then**
    return $\mathsf{Enc}_{\mathtt{k}_i}(\mathtt{r}_j)$ `// Fake if within the index`
  else
    return $\mathsf{Enc}_{\mathtt{k}_i}(\mathtt{k}_j)$ `// Real edge`
  $c_j := c_j + 1$
else
  return $\mathsf{Enc}_{\mathtt{k}_i}(\mathtt{k}_j)$ `// Real edge`
Whenever A makes a query $(\mathtt{corrupt}, v_i)$ or $(\mathtt{challenge}, v_i)$ **return** $\mathtt{k}_i$
return A's output

properties of the pebbling configurations $\mathcal{P}_0$ and $\mathcal{P}_\tau$, it is easy to see that $\mathsf{H}_{\mathsf{P},0} \equiv \mathsf{H}_{\mathsf{L}}$ and $\mathsf{H}_{\mathsf{P},\tau} \equiv \mathsf{H}_{\mathsf{R}}$. Moreover, $\mathsf{H}_{\mathsf{P},I}$ is $(t - O(t_{\mathsf{P}} + N \cdot t_{\mathcal{K}} + N^2 \cdot t_{\mathsf{Enc}}), \epsilon)$-indistinguishable from $\mathsf{H}_{\mathsf{P},I+1}$, for $I \in [0, \tau - 1]$ – we do not give the details, but the main point is to plug in the challenge ciphertext at the (only) edge that is different in the two hybrids. Thus, $\mathsf{H}_{\mathsf{P},0}, \ldots, \mathsf{H}_{\mathsf{P},\tau}$ constitutes a valid sequence of fully selective hybrids. □

Partially Selective Hybrids.

Here we show that the fully selective sequence of hybrids $\mathsf{H}_{\mathsf{P},0}, \ldots, \mathsf{H}_{\mathsf{P},\tau}$ constructed in the previous subsection is also partially selective, with each $\mathsf{H}_{\mathsf{P},I}$ of the form prescribed in Eq.3.1.

This is made possible by the fact that an ordered pebbling is, as we discussed, compressible. As a result, better bounds on adaptive security follow by Theorem 2.

Lemma 4. *For $\delta_{in} = \delta_{in}(N)$ and $D = D(N)$, let $\mathcal{G} = \mathcal{G}(N, \delta_{in}, D)$ denote the subset of key-graphs in $\mathcal{G}(N)$ with in-degree δ_{in} and depth D. For $G \in \mathcal{G}(N, \delta_{in}, D)$, let $(\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ be an ordered edge-pebbling that is generated by a deterministic edge-pebbling algorithm* P*. Let σ' denote its lateral space complexity. Then, for $I \in [0, \tau - 1]$ and $b \in \{0, 1\}$,*

$$\mathsf{H}_{\mathsf{P},I+b} = \mathrm{SEL}_{\mathcal{U}\to\mathcal{G}}[\hat{\mathsf{H}}_{I,b}, \mathsf{w}, \mathsf{h}_I],$$

where $\hat{\mathsf{H}}_{I,b}$ is defined in Algorithm 3.3, and $\mathcal{U}$ is a subset of $(\mathcal{V} \times [1, \delta_{in}])^{2\sigma'}$.

As a corollary to Lemma 4 and Theorem 2, we get our main result for the GSD game.

Theorem 3. *Let $N, \delta_{in}, \sigma', \tau$ be as in Lemma 4. Let t_{P} denote the time complexity of* P*, then an encryption scheme* (Enc, Dec) *that is (t, ϵ)-secure under IND-CPA, is $(t - O(t_{\mathsf{P}} + N \cdot t_{\mathcal{K}} + N^2 \cdot t_{\mathsf{Enc}}), \epsilon \cdot \tau \cdot (N \cdot \delta_{in})^{\sigma'})$-adaptive GSD-secure.*

Proof of lemma. The partial selectivising of $\mathsf{H}_{\mathsf{P},I+b} = \mathrm{SEL}_{\mathcal{U}\to\mathcal{G}}[\hat{\mathsf{H}}_{I,b}, \mathsf{w}, \mathsf{h}_I]$ is shown in Algorithm 3.3. The proof for the indistinguishability of $\hat{\mathsf{H}}_{I,0}$ and $\hat{\mathsf{H}}_{I,1}$ follows the same line of argument as in the proof of Lemma 3 and is omitted. For the optimized result stated in the theorem, note that the set $\mathcal{U}$ can be compressed to $(\mathcal{V} \times [1, \delta_{in}])^{\sigma'}$ since $\mathcal{Q}_I$ and $\mathcal{Q}_{I+1}$ only differ in one edge. Furthermore, to simulate the game it suffices to sample N keys. □

Corollaries.

As corollaries to Theorem 3, we capture the existing results on GSD: viz., the result on paths using the nested hybrid technique, given in [FJP15], which was briefly discussed in Section 3.2.1, and the bound for general DAGs in [Pan07].

Corollary 1 (GSD for path graphs: loss quasi-polynomial in length)**.** *Let $\mathcal{C}(N) := \mathcal{G}(N + 1, 1, N+1)$ denote the set of all chain key-graphs of N edges. An encryption scheme* (Enc, Dec) *that is (t, ϵ)-secure under IND-CPA is $(t - O(s_{\mathsf{P}_1} + N \cdot t_{\mathcal{K}} + N \cdot t_{\mathsf{Enc}}), \epsilon \cdot 3^{\log N} \cdot N^{\log N + 1})$-adaptive GSD-secure on $\mathcal{C}(N)$.*

Proof. Suppose that N is a power of two. An edge-pebbling algorithm P_1 for chain graphs $C(N) \in \mathcal{C}(N)$ is given in Algorithm 3.4 – the algorithm is to be invoked on $((0, N), (0, N))$ in order to pebble the sink. It is called the nested pebbling strategy as it is used implicitly in the argument for path graphs using nested hybrids in [FJP15]. The number of pebbles used by P_1 is captured by the recursion $\sigma_{\mathsf{P}_1}(C(N)) = \sigma_{\mathsf{P}_1}(C(N/2)) + 1$, with $\sigma_{\mathsf{P}_1}(C(1)) = 1$.[12] The number of moves, on the other hand, is captured by $\tau_{\mathsf{P}_1}(C(N)) = 3 \cdot \tau_{\mathsf{P}_1}(C(N/2))$ with $\tau_{\mathsf{P}_1}(C(1)) = 1$. Therefore, $\sigma_{\mathsf{P}_1}(\mathcal{C}(N)) = \log N + 1$ and $\tau_{\mathsf{P}_1}(\mathcal{C}(N)) = 3^{\log N}$. As the indegree of a path graph is one, all its pebbling sequences are ordered, and its lateral space complexity is the same as the number of pebbles, i.e. $\sigma' = \log N + 1$. The corollary now follows from Theorem 3. A similar argument can be made for arbitrary N. □

[12] It is known that the strategy is optimal – i.e., $\sigma(C(N)) = \sigma_{\mathsf{P}_1}(C(N))$ [Krá01]. We will outline a proof for this statement in 7.5.1 in Chapter 7.

Algorithm 3.3: Partially selectivized hybrids. $\mathsf{H}_{\mathsf{P},I+b} := \mathrm{SEL}_{\mathcal{U}\to\mathcal{G}}[\hat{\mathsf{H}}_{I,b}, \mathsf{w}, \mathsf{h}_I]$, where $\mathcal{U}$ is a subset of $(\mathcal{V} \times [1, \delta_{in}])^{2\sigma'}$.

$\mathsf{H}^{\mathsf{A}}_{\mathsf{P},I}$
1 Obtain the key-graph $G \in \mathcal{G}$ from A
2 Compute $\mathcal{P}_0, \ldots, \mathcal{P}_\tau \leftarrow \mathsf{P}(G)$
3 Compute $\mathcal{Q}_I$, $\mathcal{Q}_{I+1}$
4 Run $\hat{\mathsf{H}}_{I,b}(\mathcal{Q}_I, \mathcal{Q}_{I+1})$
5 **return** $\hat{\mathsf{H}}_{I,b}$'s output

$\hat{\mathsf{H}}^{\mathsf{A}}_{I,b}(\mathcal{Q}_I, \mathcal{Q}_{I+1})$
6 Initialise $c_1, \ldots, c_N := 0$
7 Choose $2N$ keys $\mathtt{r}_1, \ldots, \mathtt{r}_N, \mathtt{k}_1, \ldots, \mathtt{k}_N \leftarrow \mathcal{K}$
8 Whenever A makes a query $(\mathtt{encrypt}, i, j)$:
9 **if** $(v_j \in \mathcal{Q}_{I+b})$ **then**
10 | **if** $(c_j \leq \mathit{index}_j \in \mathcal{Q}_{I+b})$ **then** `// The index of` v_j `in` $\mathcal{Q}_I$ `is compared`
11 | | **return** $\mathsf{Enc}_{\mathtt{k}_i}(\mathtt{r}_j)$
12 | **else**
13 | | **return** $\mathsf{Enc}_{\mathtt{k}_i}(\mathtt{k}_j)$
14 | $c_j := c_j + 1$
15 **else**
16 | **return** $\mathsf{Enc}_{\mathtt{k}_i}(\mathtt{k}_j)$
17 Whenever A makes a query $(\mathtt{corrupt}, i)$ or $(\mathtt{challenge}, i)$: **return** $\mathtt{k}_i$
18 **return** A's output

Corollary 2 (GSD for DAGs: loss exponential in depth)**.** *For $D = D(N)$, let $\mathcal{G}(N, D)$ denote the subset of graphs in $\mathcal{G}(N)$ with depth D. An encryption scheme* (Enc, Dec) *that is (t, ϵ)-secure under IND-CPA is $(t - O(t_{\mathsf{P}_2} + N \cdot t_{\mathcal{K}} + N^2 \cdot t_{\mathsf{Enc}}), \epsilon \cdot (2N)^D \cdot N^{2D})$-adaptive GSD-secure on $\mathcal{G}(N, \delta_{in}, D)$.* [13]

Proof. A reversible edge-pebbling strategy that pebbles a single edge (u^*, v^*) is given in Algorithm 3.5. In order to pebble a vertex v^*, the algorithm is to be called sequentially on all $u^* \in \mathsf{parents}(v^*)$. An example of this pebbling strategy is given in Figure 3.8. Let's first see why the generated pebbling sequence is ordered. For an edge (u, v), the order in which the edges incident on a vertex u are pebbled is determined by $\mathsf{parents}(u)$, which we assumed preserves the edge ordering. Moreover, the edges are restored in the opposite order. Together, these two steps ensure that in any configuration, if an edge (u, v) carries a pebble then all the edges that are incident on v and *precede* (u, v) must also carry pebbles. In addition, all the vertices that have edges with pebbles coming into them lie along a path from a source to a sink – it follows that the lateral edge-pebbling complexity σ' is at most D. The number of moves in the above strategy is captured by the recursion $T(D) \leq 2N \cdot T(D-1)$ with $T(1) \leq 2N$, and hence $\tau_{\mathsf{P}_2}(\mathsf{G}(N, D)) < (2N)^D$. Plugging in these values in Theorem 3 proves the corollary. □

[13] While asymptotically we get a similar factor $N^{O(D)}$ loss in security as [Pan07], recall that Panjwani obtained a stronger concrete bound $(2N)^{D+1}$. For small indegree δ_{in}, however, we get a comparable bound $(2\delta_{in})^D \cdot (N\delta_{in})^D$.

Algorithm 3.4: Nested pebbling for path graphs with N edges, where N is a power of two. Note that the initial call must be $\mathsf{P}_1((0,N),(0,N))$.

```
P_1((A,B),(a,b))
if b = a + 1 then
    if (a, a+1) is pebbled then
        remove it                                   // P_{i+1} := P_i \ {(a, a+1)}
    else
        place pebble on (a, a+1)                    // P_{i+1} := P_i ∪ {(a, a+1)}
                                                    // Increment counter i
else
    P_1((A,B),(a,(a+b)/2))                          // Recursively pebble left half
    P_1((A,B),((a+b)/2,b))                          // Recursively pebble right half
    P_1((A,B),(a,(a+b)/2)))                         // Recursively unpebble left half
if (A,B) = (a,b) then
    HALT
```

Algorithm 3.5: A recursive edge-pebbling algorithm. Note that to pebble an edge $(u,v) \in \mathcal{E}$, the initial call must be $\mathsf{P}_2(G,(u,v),(u,v))$

```
P_2(G,(u*,v*),(u,v))
for x ∈ parents(u) do
    P_2(G,(u*,v*),(x,u))                            // Pebble parents recursively
if (u,v) is pebbled then
    remove pebble on (u,v)                          // P_{i+1} := P_i \ {(u,v)}
else
    place pebble on (u,v)                           // P_{i+1} := P_i ∪ {(u,v)}
                                                    // Increment counter i
for x ∈ parents^{-1}(u) do
    P_2(G,(u*,v*),(x,u))                            // Unpebble parents, in reverse
if (u,v) = (u*,v*) then
    HALT
```

3.4 Application II: Constrained Pseudorandom Functions

In this section, we formalise the high level ideas that we presented in Section 3.1.2 and reprove the results from [FKPR14] on adaptive security of the prefix-constrained PRF from [GGM84] using our framework.

3.4.1 Formal Definitions

The formal definitions are essentially taken from [FKPR14]. First, we recall the definition of pseudorandom generators.

Definition 11. An efficient function $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ is a (t,ϵ)-secure (length-doubling) *pseudorandom generator* (PRG) if

$$\mathsf{PRG}(U_\lambda) \approx_{(t,\epsilon)} U_{2\lambda}.$$

The GGM pseudorandom function (PRF) is now defined based on a length-doubling PRG.

Definition 12 (GGM PRF)**.** Given a pseudorandom generator $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$, the PRF $\mathsf{F}_{\mathsf{GGM}} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$ is defined as

$$\mathsf{F}_{\mathsf{GGM}}(\mathrm{k}, x) = \mathrm{k}_x \text{ where } \mathrm{k}_\emptyset = \mathrm{k} \text{ and } \forall z \in \{0,1\}^* : \mathrm{k}_{z\|0} \| \mathrm{k}_{z\|1} = \mathsf{PRG}(\mathrm{k}_z).$$

Next, we give the definitions for cPRFs that are tailored to prefix-constrained PRFs.

Definition 13 (Prefix-constrained PRF)**.** For $n \in \mathbb{N}$, a function $\mathsf{F} : \mathcal{K} \times \{0,1\}^n \to \mathcal{Y}$ is a *prefix-constrained PRF* if there are algorithms $\mathsf{F.Constrain} : \mathcal{K} \times \{0,1\}^{\leq n} \to \mathcal{K}_{pre}$ and $\mathsf{F.Eval} : \mathcal{K}_{pre} \times \{0,1\}^n \to \mathcal{Y}$ which for all $\mathrm{k} \in \mathcal{K}$, $x \in \{0,1\}^{\leq n}$ and $\mathrm{k}_x \leftarrow \mathsf{F.Constrain}(\mathrm{k}, x)$ satisfy

$$\mathsf{F.Eval}(k_x, x') = \begin{cases} \mathsf{F}(\mathrm{k}, x') & \text{if } x \text{ is a prefix of } x' \\ \perp & \text{otherwise.} \end{cases}$$

That is, $\mathsf{F.Constrain}(\mathrm{k}, x)$ outputs a key k_x that allows evaluation of $\mathsf{F}(\mathrm{k}, \cdot)$ on all inputs that have x as a prefix. We can derive a prefix-constrained PRF from the GGM construction by setting $\mathcal{K} = \{0,1\}^\lambda$, $\mathcal{Y} = \{0,1\}^\lambda$, and for a random $\mathrm{k} \leftarrow \mathcal{K}$ and $x \in \{0,1\}^l$ with $l \leq n$ defining $\mathsf{F}_{\mathsf{GGM}}.\mathsf{Constrain}(\mathrm{k}, x) = (\mathrm{k}_x^{(1)}, \mathrm{k}_x^{(2)}) := (x, \mathsf{F}_{\mathsf{GGM}}(\mathrm{k}, x))$ and

$$\mathsf{F}_{\mathsf{GGM}}.\mathsf{Eval}(\mathrm{k}_x, x') := \begin{cases} \mathsf{F}_{\mathsf{GGM}}(\mathrm{k}_x^{(2)}, z) & \text{if } x' = x||z \text{ for some } z \in \{0,1\}^{n-l} \\ \perp & \text{otherwise.} \end{cases}$$

The security for prefix-constrained PRFs is argued using the following game.

Definition 14. The game is played between a challenger G (which is either G_L or G_R) and an adversary A using F. The challenger G picks a random key $\mathrm{k} \leftarrow \mathcal{K}$, and initialises a set $\mathcal{X} = \emptyset$. A can make at most $q = q(n)$ queries, which is either:

- Constrain queries, $(\mathtt{constrain}, x)$: G returns $\mathsf{F.Constrain}(\mathrm{k}, x)$, and adds x to $\mathcal{X}$.
- One challenge query $(\mathtt{challenge}, x^*)$: Here the answer differs between G_L and G_R: G_L answers with $\mathsf{F.Eval}(\mathrm{k}, x^*)$ (real output), whereas G_R answers with random $r \leftarrow \mathcal{Y}$ (fake, random output) – for the task to be non-trivial, no element in $\mathcal{X}$ must be a prefix of x^*. G adds x^* to $\mathcal{X}$.

In the fully selectivized version of Definition 14, A must commit to the whole set $\mathcal{X} := \{x_1, \ldots, x_q\}$. Therefore, the selective challenger is defined as $\mathsf{H}_\mathsf{L} = \mathrm{SEL}_{\{0,1\}^{n \cdot q}}[\mathsf{G}_\mathsf{L}, \mathsf{w}]$ (resp., $\mathsf{H}_\mathsf{R} = \mathrm{SEL}_{\{0,1\}^{n \cdot q}}[\mathsf{G}_\mathsf{R}, \mathsf{w}]$), where w is the function that extracts $\mathcal{X}$ from the transcript. Note that the amount of information that A commits to in the selectivised games is much more than the one defined in [FKPR14].

Definition 15. A prefix-constrained PRF F is (t, ϵ, q)-adaptive-secure (resp., selective-secure) if G_L and G_R (resp., H_L and H_R) are (t, ϵ)-indistinguishable.

3.4.2 Hybrids and Pebbling Configurations

Fully Selective Hybrids.

Let's briefly recall from Section 3.1.2 how we used the knowledge of $x_1, \ldots, x_q$ to get a better sequence of hybrids. First we switched to the recursive pebbling sequence in Figure 3.1.b.

Algorithm 3.6: Template for generating fully selective hybrids. The sub-routine $\mathsf{K}(x)$ computes the key k_x from the first key that has already been defined on the path from x to the root.

$\mathsf{H}^{\mathsf{A}}(\mathcal{P}_I)$

Obtain $\mathcal{X} \in \{0,1\}^{n \cdot q}$ from A
Compute $\mathbf{index} := \mathsf{index}(\mathcal{X}, \mathcal{P}_I)$
Sample key $\mathrm{k}_\emptyset \leftarrow \mathcal{K}$, set $\forall\, \emptyset \neq x \in \{0,1\}^{\leq n} : \mathrm{k}_x := \perp$ `// Initialise the keys`
Initialise the counter $c = 1$
Whenever A makes a query $(\mathtt{constrain}, x)$:
  if $c = \mathit{index}_i \in \mathbf{index}$ **then** set $\mathrm{k}_{x[1,i]\|0} \| \mathrm{k}_{x[1,i]\|1} \leftarrow U_{2\lambda}$ `// Fake output`
  Increment c by one
  return $\mathsf{K}(x)$ `// Compute the key using the sub-routine`
return A's output

$\mathsf{K}(x)$

if $k_x \neq \perp$ **then return** k_x `// Key already defined`
Set $l = |x| - 1$, $\mathrm{k}_{x[1,l]} = \mathsf{K}(x[1,l])$ `// Recursively compute the key`
$\mathrm{k}_{x[1,l]\|0} \| \mathrm{k}_{x[1,l]\|1} := \mathsf{PRG}(\mathrm{k}_{x[1,l]})$ `// Normal output`
return k_x

Second, we managed to shrink the index of a pebble from $[0, 2^n]$ to $[1, q]$ by assuming an upper bound q on the number of queries made by the adversary: we set the index of a pebble to the index of the first constrain query whose i bit prefix coincides with x^*. More formally, the index for a pebble on the ith edge is defined as

$$\mathsf{index}(\mathcal{X}, i) := \arg \min_{j \in [1,q]} \{x^*[1,i] = x_j[1,i]\}.$$

The index of an edge-pebbling configuration is accordingly defined as

$$\mathsf{index}(\mathcal{X}, \mathcal{P}_I) := \{\mathsf{index}_i : (i, i+1) \in \mathcal{P}_I, \mathsf{index}_i = \mathsf{index}(\mathcal{X}, i)\}.$$

By using the edge-pebbling $\mathcal{P}_0, \ldots, \mathcal{P}_\tau$ generated by P_1 (Algorithm 3.4), where $\tau = 3^{\log n}$, we get a sequence of fully selective hybrids $\mathsf{H}(\mathcal{P}_0), \ldots, \mathsf{H}(\mathcal{P}_\tau)$, with $\mathsf{H}(\mathcal{P}_I)$ described in Algorithm 3.6, and the following lemma.

Lemma 5. *Let $\tau := 3^{\log n}$. The sequence of hybrids $\mathsf{H}(\mathcal{P}_0), \ldots, \mathsf{H}(\mathcal{P}_\tau)$ constitutes a valid sequence of fully selective hybrids. Furthermore, if the PRG is (t, ϵ)-indistinguishable then the constrained PRF $\mathsf{F}_{\mathsf{GGM}}$ is $(t - O(t_{\mathsf{P}_1} + qn \cdot t_{\mathsf{PRG}}), \epsilon \cdot \tau, q)$-selective secure, where t_{P_1} (resp, t_{PRG}) denotes the time complexity of P_1 (resp., PRG).*

Proof. By the properties of the pebbling configurations $\mathcal{P}_0$ and $\mathcal{P}_\tau$, it is easy to see that $\mathsf{H}(\mathcal{P}_0) \equiv \mathsf{H}_{\mathsf{L}}$ an $\mathsf{H}(\mathcal{P}_\tau) \equiv \mathsf{H}_{\mathsf{R}}$. In addition, the neighbouring hybrids $\mathsf{H}(\mathcal{P}_I)$ and $\mathsf{H}(\mathcal{P}_{I+1})$ are $(t - qn \cdot t_{\mathsf{PRG}}, \epsilon, q)$-indistinguishable – see Algorithm 3.7 for a reduction. The lemma follows. □

Lemma 6. *The sequence of fully selective hybrids $\mathsf{H}(\mathcal{P}_0), \ldots, \mathsf{H}(\mathcal{P}_{3^{\log n}})$ are partially selective as*

$$\mathsf{H}(\mathcal{P}_{I+b}) = \mathrm{SEL}_{\{0,1\}^{2 \cdot \log n \cdot \log q} \to \{0,1\}^{n \cdot q}}[\hat{\mathsf{H}}_{I,b}, \mathsf{w}, \mathsf{h}_I], \tag{3.7}$$

Algorithm 3.7: The reduction algorithm that establishes the indistinguishability of $\mathsf{H}(\mathcal{P}_I)$ and $\mathsf{H}(\mathcal{P}_{I+1})$. (See Algorithm 3.6 for the description of K.)

$\mathsf{R}^{\mathsf{A}}(\mathcal{P}_I, \mathcal{P}_{I+1}, y)$ // $y \in \{0,1\}^{2\lambda}$ is the PRG challenge

Obtain $\mathcal{X} \in \{0,1\}^{n \cdot q}$ from A

$\mathbf{index} := \mathsf{index}(\mathcal{X}, \mathcal{P}_I \cap \mathcal{P}_{I+1})$, $\mathsf{index}_{i^*} := \mathsf{index}(\mathcal{X}, \mathcal{P}_I \triangle \mathcal{P}_{I+1})$

Sample key $\mathtt{k}_\emptyset \leftarrow \mathcal{K}$, set $\forall\, \emptyset \neq x \in \{0,1\}^{\leq n} : \mathtt{k}_x := \perp$ // Initialise the keys

Initialise the counter $c = 1$

Whenever A makes a query $(\mathtt{constrain}, x)$:

  if $c = \mathit{index}_{i^*}$ **then** set $\mathtt{k}_{x[1,i^*]\|0} \| \mathtt{k}_{x[1,i^*]\|1} := y$ // Fake or real

  if $c = \mathit{index}_i \in \mathbf{index}$ **then** set $\mathtt{k}_{x[1,i]\|0} \| \mathtt{k}_{x[1,i]\|1} \leftarrow U_{2\lambda}$ // Fake output

  Increment c by one

  return $\mathsf{K}(x)$ // Compute the key using the sub-routine

return A's output

Algorithm 3.8: Partially selectivized hybrids $\hat{\mathsf{H}}_{I,b}$ for $\mathsf{H}_{I+b} := \mathsf{H}(\mathcal{P}_{I+b})$. (See Algorithm 3.6 for the description of K.)

$\hat{\mathsf{H}}^{\mathsf{A}}_{I,b}(\mathbf{index}_{I+b})$

Sample key $\mathtt{k}_\emptyset \leftarrow \mathcal{K}$, set $\forall\, \emptyset \neq x \in \{0,1\}^{\leq n} : \mathtt{k}_x := \perp$ // Initialise the keys

Initialise the counter $c = 1$

Whenever A makes a query $(\mathtt{constrain}, x)$:

  if $c = \mathit{index}_i \in \mathbf{index}_{I+b}$ **then** set $\mathtt{k}_{x[1,i]\|0} \| \mathtt{k}_{x[1,i]\|1} \leftarrow U_{2\lambda}$ // Faked

  Increment c by one

  return $\mathsf{K}(x)$ // Compute the key using the sub-routine

return A's output

where $\hat{\mathsf{H}}_{I,b}$ *is described in Algorithm 3.8, and* h_I *is the function that, on input the queries* $\mathcal{X}$*, computes the indices* $\mathbf{index}_I$ *and* $\mathbf{index}_{I+1}$ *of the pebbling configurations* $\mathcal{P}_I$ *and* $\mathcal{P}_{I+1}$*, respectively.*[14]

It follows from Theorem 2 and Lemma 6 that the prefix-constrained PRF $\mathsf{F}_{\mathsf{GGM}}$ is adaptive-secure with a quasi-polynomial loss in tightness.

Theorem 4. *If the underlying PRG is* (t, ϵ)*-indistinguishable then* $\mathsf{F}_{\mathsf{GGM}}$ *is a* $(t - O(t_{\mathsf{P}_1} + qn \cdot t_{\mathsf{PRG}}), \epsilon \cdot 3^{\log n} \cdot n^{2 \cdot \log q})$*-adaptive-secure prefix-constrained PRF.*

Proof of Lemma 6. It remains to show that $\hat{\mathsf{H}}_{I,0}$ and $\hat{\mathsf{H}}_{I,1}$ are indistinguishable – the reduction is given in Algorithm 3.9. The lemma follows. □

[14] There are means to further compress h_I: it suffices that it returns the indices $\mathsf{index}(\mathcal{X}, \mathcal{P}_I \cap \mathcal{P}_{I+1})$ and $\mathsf{index}(\mathcal{X}, \mathcal{P}_I \triangle \mathcal{P}_{I+1})$, along with a bit b^* which indicates what the pebbling move is – i.e., if, in move $I+1$, the edge (i^*, i^*+1) was added to $\mathcal{P}_I$ then $b^* = 0$, otherwise $b^* = 1$. But we prefer the simpler h_I for the sake of simplicity of exposition.

Algorithm 3.9: The reduction algorithm that establishes the indistinguishability of the partially selectivized hybrids $\hat{\mathsf{H}}_{I,0}$ and $\hat{\mathsf{H}}_{I,1}$. (See Algorithm 3.6 for the description of K.)

$\mathsf{R}'^{\mathsf{A}}(\mathbf{index}_I, \mathbf{index}_{I+1}, y)$ // $y \in \{0,1\}^{2\lambda}$ is the PRG challenge

Let $\mathsf{index}_{i^*} := \mathbf{index}_I \triangle \mathbf{index}_{I+1}$, $\mathbf{index} := \mathbf{index}_I \cap \mathbf{index}_{I+1}$

Sample key $\mathrm{k}_\emptyset \leftarrow \mathcal{K}$, set $\forall\, \emptyset \neq x \in \{0,1\}^{\leq n} : \mathrm{k}_x := \perp$ // Initialise the keys

Initialise the counter $c = 1$

Whenever A makes a query $(\mathtt{constrain}, x)$:

  if $c = \mathit{index}_{i^*}$ **then** set $\mathrm{k}_{x[1,i^*]\|0} \| \mathrm{k}_{x[1,i^*]\|1} := y$ // Fake or real output

  if $c = \mathit{index}_i \in \mathbf{index}$ **then** set $\mathrm{k}_{x[1,i]\|0} \| \mathrm{k}_{x[1,i]\|1} \leftarrow U_{2\lambda}$ // Fake output

  Increment c by one

  return $\mathsf{K}(x)$ // Compute the key using the sub-routine

return A's output

3.5 Open Problems

In this chapter we presented a framework for proving adaptive security of various schemes including generalized selective decryption, constrained PRFs, and Yao's garbled circuits. Further applications can be found in our publication [JKK+17a] (secret sharing over access structures defined via monotone circuits), the work of Kowalczyk and Wee [KW19] (attribute-based encryption), and the following Chapters of this thesis (proxy re-encryption, group key agreement, Yao's garbling).

In all these applications of the framework, the security loss of a scheme is captured by the existence of some pebbling strategy. Thus, it is natural to ask the following questions: Does there exist a connection in the opposite direction between the security loss of a scheme and possible pebbling strategies? That is, is it possible to use lower bounds for pebbling strategies to show that various security losses are necessary? In Chapters 7 and 8 we give a positive answer to these questions by providing first lower bounds on the security loss of various schemes.

Part III

Upper Bounds

CHAPTER 4

Adaptively Secure Proxy Re-encryption

4.1 Introduction

A proxy re-encryption (PRE) scheme is a public-key encryption scheme with an additional functionality: Alice and Bob, who have key pairs $(\mathtt{pk}_A, \mathtt{sk}_A)$ and $(\mathtt{pk}_B, \mathtt{sk}_B)$, respectively, can generate a re-encryption key (re-key, for short) $\mathtt{rk}_{A,B}$ that allows its holder, say Peggy, to act as a proxy; that is, she can transform ciphertexts under $\mathtt{pk}_A$ to ciphertexts under $\mathtt{pk}_B$ without having to know the underlying message. A trivial way to accomplish this would be for Alice to hand her secret key $\mathtt{sk}_A$ to Peggy, who can then decrypt ciphertexts under $\mathtt{pk}_A$, encrypt them under $\mathtt{pk}_B$ and send them to Bob. Alice's secret key acts thus as the re-key and de- and encryption algorithms are used for re-encryption. However, this approach requires Alice to reveal her secret key to Peggy and therefore place complete trust on her. The more interesting cases are when the parties are mutually distrustful.

Bidirectional vs. unidirectional. In the above setting, if the re-key $\mathtt{rk}_{A,B}$ allows Peggy to also transform ciphertexts under $\mathtt{pk}_B$ to $\mathtt{pk}_A$, the PRE scheme is called "bidirectional". For such schemes the re-key is necessarily a function of both $\mathtt{sk}_A$ and $\mathtt{sk}_B$. In this paper we are interested in the more interesting case of "unidirectional" PRE schemes where the re-key $\mathtt{rk}_{A,B}$ can only transform ciphertexts from $\mathtt{pk}_A$ to $\mathtt{pk}_B$, and not vice-versa, and ciphertexts under $\mathtt{pk}_B$ remain secure even given $\mathtt{sk}_A$ and $\mathtt{rk}_{A,B}$. (Henceforth we will always assume PRE to be unidirectional.) As opposed to bidirectional PRE, the re-key generation algorithm in a unidirectional PRE takes as input "source" keys $(\mathtt{pk}_A, \mathtt{sk}_A)$ and only the "target" public key $\mathtt{pk}_B$.

Single hop vs. multiple hops. Suppose a third user, Charlie, holding keys $(\mathtt{pk}_C, \mathtt{sk}_C)$, enters the picture and suppose Peggy obtains the re-key $\mathtt{rk}_{B,C}$ that allows her to transform ciphertexts under Bob's public key to ciphertexts under Charlie's public key. Peggy can, by definition, transform a ciphertext $\mathtt{c}_A$ under $\mathtt{pk}_A$ to a ciphertext $\mathtt{c}_B$ under $\mathtt{pk}_B$ using her re-key $\mathtt{rk}_{A,B}$. If the scheme allows Peggy to transform a ciphertext $\mathtt{c}_B$, which has already been re-encrypted once, to a ciphertext $\mathtt{c}_C$ under $\mathtt{pk}_C$ using the re-key $\mathtt{rk}_{B,C}$ then we say that the PRE scheme allows two "hops". In a similar manner, one can consider multiple hops

This Chapter essentially replicates, with permission, the full version [FKKP18] of our publication [FKKP19], © IACR 2019, https://doi.org/10.1007/978-3-030-17259-6_11.

of re-encryptions. Such a scheme is termed "multi-hop" as opposed to a "single-hop" scheme (which does not allow re-encryptions of already re-encrypted ciphertexts).

4.1.1 Modelling Security

The basic notion of security for unidirectional PRE is that of indistinguishability under chosen-plaintext attack (CPA). There are N users and, at the beginning of the game, the adversary gets their public keys $\mathtt{pk}_1, \ldots, \mathtt{pk}_N$ from the challenger. In the first phase, the adversary can corrupt users of its choice by requesting their secret keys; in the second phase, it can obtain re-keys $\mathtt{rk}_{i,j}$ and re-encryptions for ciphertexts of its choice. The scheme is CPA-secure if it is infeasible for the adversary to distinguish encryptions of two messages under a key that the adversary has not corrupted either directly or indirectly (through a re-key or re-encryption query to a corrupted user).

Just as in standard public-key encryption, the above security definition can be strengthened to chosen-ciphertext attack (CCA) by allowing the adversary access to a decryption oracle which, on input a ciphertext and a public key $\mathtt{pk}_i$ returns the decryption of the ciphertext under $\mathtt{sk}_i$. The conditions to ensure non-triviality have to be altered accordingly.

We note that both definitions are *selective* in nature: the adversary must choose the set of players it corrupts before issuing any queries.

4.1.2 Prior Work

Bidirectional PRE was introduced as "atomic proxy cryptography" by Blaze, Bleumer and Strauss [BBS98a], who constructed a multi-hop scheme under the decisional Diffie-Hellman assumption. Unidirectional PRE was introduced later by Ateniese et al. [AFGH05]. Their main motivation was to limit the amount of trust placed on the proxy, as required by their application to access control for distributed storage. Since the notion of security for unidirectional PRE is different from bidirectional PRE, they also reformulated the notion of CPA (for the single-hop setting). Assuming hardness of certain problems on bilinear groups, they constructed CPA-secure schemes that are single-hop and unidirectional.

The definition of CCA security for single-hop bidirectional schemes is due to Canetti and Hohenberger [CH07] and is more involved than previous definitions, mainly because the adversary is allowed adaptive corruption. They gave a scheme satisfying their notion under the standard decisional bilinear Diffie-Hellman assumption. The definition of CCA security in the unidirectional setting is due to Libert and Vergnaud [LV08], who instantiate it under a slightly non-standard assumption on bilinear groups.

The earlier constructions of multi-hop, unidirectional schemes were based on program obfuscation [HRsV07, CCV12]. In his seminal paper, Gentry [Gen09] gave a generic construction of PRE from fully homomorphic encryption. The first construction (with succinct ciphertexts) based on a standard assumption is due to Chandran et al. [CCL+14]: their scheme is CPA-secure assuming decisional learning with errors. Phong et al. [PWA+16] followed up with a construction that, in addition, enjoys a security property called "key-privacy". The only construction of a CCA-secure multi-hop, unidirectional scheme is due to Fan and Liu [FL19]. In their paper, they also defined the security models (CPA and CCA) for the multi-hop setting.

Cohen [Coh19] has recently argued that CPA security might be too weak for some applications and introduced *indistinguishability against honest-reencryption attack* (HRA), a notion that

implies CPA (but is incomparable to CCA). He also showed that if a PRE scheme satisfies a property called "source-hiding", which several existing CPA-secure schemes do, then HRA security follows from CPA security.

4.1.3 Our Contribution

Our starting point is the observation that, unlike bidirectional PRE, the security definitions for unidirectional PRE (that is, CPA, HRA and CCA) are all *selective* in nature: the adversary must choose the set of parties it corrupts before issuing any queries. A more meaningful notion would be *adaptive* security, where the adversary is allowed to corrupt users at any time during the game. However, modelling this turns out to be as tricky as in the bidirectional setting. In this paper, we lift the definitions for CPA and HRA to the adaptive setting.

First Contribution: Modelling Adaptive Corruption.

The main problem that arises when we allow the adversary to adaptively corrupt users is that we must ensure that the adversary cannot trivially win the security game. For bidirectional PRE this was handled in [CH07] by defining a relation that keeps track of the dependency between the re-keys and re-encryptions that were issued during the game. Our approach is similar in spirit: the security game maintains a "recoding graph" that has N nodes, and whose edges are derived from the re-keys and re-encryptions issued to the adversary. The exact definitions of the recoding graph for adaptive CPA and for adaptive HRA differ slightly, but in both cases it is defined so that no corrupt key is reachable from the challenge key. That is, the adversary is forbidden from making *any* re-key or re-encryption queries from a key that is reachable from the challenge key to a corrupt user. The recoding graph now allows to ensure non-triviality of the adversary's actions by checking a few basic graph properties.

Second Contribution: The Reduction.

Proving adaptive security can be reduced to showing selective security by initially guessing the set of users that will be corrupted. However, this reduction loses an exponential factor in N, rendering the reduction meaningless already for moderate N. As our main contribution, we give a more fine-grained reduction from adaptive to selective security which in many practical settings and for several existing schemes (or minor variants thereof) implies adaptive security at much smaller (quasi-polynomial, or even polynomial) loss. More precisely, the loss in our reduction depends on the structure of the recoding graph: for trees and chains we get a quasi-polynomial $N^{O(\log N)}$ loss, whereas for general graphs the loss is exponential in their depth. Fortunately, trees, chains, and low-depth graphs cover many, if not most, interesting applications.

Security assumptions. A key step in our search for a tighter reduction was the identification of the basic security assumptions on a PRE scheme that we required in our arguments. For the case of CPA, it turned out to be ciphertext indistinguishability and *weak* key-privacy, both fairly standard security requirements already explored in some of the previous works.

As the name suggests, a PRE scheme is ciphertext-indistinguishable (or, for short, indistinguishable) if the underlying encryption is. Since the syntax of the encryption algorithm for a PRE scheme is slightly different from that of a standard public-key encryption, the definition of indistinguishability has to be slightly changed. To be precise, the encryption algorithm for a PRE scheme takes also a "level" as input, and we require that the ciphertexts are

indistinguishable *on all levels*. It is not hard to see that any selectively CPA-secure PRE scheme has to trivially satisfy indistinguishability.

The notion of key-privacy was introduced in a strong form in [ABH09]. We require the PRE scheme to satisfy a much weaker property: While strong key-privacy requires that a re-key $\mathrm{rk}_{A,B}$ looks pseudorandom given just the source and target public keys pk_A and pk_B, weak key-privacy only requires a re-key to hide the source key. Existing PRE schemes that satisfy the stronger key privacy as defined in [ABH09] are therefore candidates for our reduction.

To apply our reduction to HRA-secure PRE, we need a third assumption to hold: source-hiding. This is the same property that allowed Cohen [Coh19] to lift a CPA-secure PRE scheme to an HRA-secure one. Informally, a PRE scheme is source-hiding if ciphertexts that result from re-encryptions are distributed close to fresh encryptions (at the corresponding level).

For PRE schemes satisfying these assumptions, we show that the Piecewise-Guessing framework from Chapter 3 can be applied. This is the first application of the framework in the *public-key* setting.

Applying the Piecewise-Guessing framework. First, we consider a *fully selective* security notion defined as two games H_L and H_R being indistinguishable where the adversary has to commit to *all* its choices $w \in \mathcal{W}$ (not only the set of corrupt users) right in the beginning of the game. If H_L and H_R can be proven indistinguishable via a sequence of hybrid games $(\mathsf{H}_0, \ldots, \mathsf{H}_\tau)$ with $\mathsf{H}_0 = \mathsf{H}_\mathsf{L}$ and $\mathsf{H}_\tau = \mathsf{H}_\mathsf{R}$, then adaptive security can be proven by defining a new reduction that guesses the adversary's choices $w \in \mathcal{W}$ at random and then follows the selective reduction. Now, recall that the key observation in Chapter 3 was that in many such selective reductions, only a highly compressed version $h(w)$ of the information $w \in \mathcal{W}$ that the adversary commits to is actually used in the simulation of intermediate hybrids. We called these "partially selective" hybrids, as opposed to the original hybrids, which are "fully selective", and showed that the security loss is only exponential in the length of $h(w)$ (its the longest value for any two consecutive hybrids), and not exponential in the length of the entire w.

The adversary's choices in adaptive CPA and HRA are precisely captured by the recoding graph. (Strictly speaking, it suffices to consider the subgraph that is reachable from the challenge vertex, which we will call the "challenge graph".) Similar to the case of GSD in Section 3.3 we define the hybrid games $(\mathsf{H}_0, \ldots, \mathsf{H}_\tau)$ via a pebbling sequence $(\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ following appropriately defined pebbling rules. The presence (or not) of a pebble on a vertex dictates how the re-encryption and re-key queries outgoing from that vertex are simulated. Therefore in the fully selective games, the adversary commits to the recoding graph (which is different from the original selective game in which the adversary committed to the set of corrupt users), whereas in the partially selective games it "commits" just to a pebbling configuration.

Let us first consider adaptive CPA: the edges of the recoding graph correspond to the re-key and re-encryption queries made by the adversary during the game. For simplicity, assume that the recoding graph has a single source vertex i^* that is also the vertex the adversary wants to be challenged on. When making a challenge query, the adversary receives an encryption of either m_0^* or m_1^* under pk_{i^*}; let $\mathsf{G}_\mathsf{L} = \mathsf{CPA}^0$ and $\mathsf{G}_\mathsf{R} = \mathsf{CPA}^1$ denote the respective games. In case there are no outgoing edges from i^*, indistinguishability of CPA^0 and CPA^1 follows from ciphertext indistinguishability (the first assumption): The reduction embeds the challenge public key (of the indistinguishability game) as the i^*th key, relays $(\mathrm{m}_0^*, \mathrm{m}_1^*)$ to its challenger

and forwards the challenge ciphertext it receives to the adversary. As there are no outgoing re-keys from i^*, the simulation does not require the secret key $\mathtt{sk}_{i^*}$.

In case i^* does have outgoing edges, the idea is to use a sequence of hybrids to reach a game where knowledge of $\mathtt{sk}_{i^*}$ is not required for simulation, just like above. To argue indistinguishability of hybrids, we use weak key-privacy, which guarantees that a re-key looks pseudorandom given the source and target public keys. Weak key-privacy allows the simulator to fake the outgoing edges from a vertex, after which the secret key for this vertex is not required for simulation anymore. However, the simulator cannot fake edges right away: it has to fake all children of a vertex first, before it can rely on weak key-privacy. Consequently, the pebbling must obey the following rule: in a move, a pebble can be placed on or removed from a vertex only if all its children carry pebbles.

To be precise, in game H_ℓ, for each pebbled vertex in $\mathcal{P}_\ell$ all queried re-keys outgoing from that vertex are faked. Observe that as the secret key corresponding to a vertex is used only for the generation of the re-keys outgoing from that vertex, the simulation of a hybrid can be carried out *without* knowledge of the secret keys corresponding to the pebbled vertices.

Main result. Our main result bounds the security loss for arbitrary recoding graphs in terms of their space and time complexity, where a graph is said to have space complexity σ and time complexity τ if there exists a valid pebbling strategy for that graph that uses at most σ pebbles and requires at most τ moves. More generally, a class of graphs has space complexity σ and time complexity τ if this is the case for every graph in that class.

Theorem 5 (Informal Theorem 7 and Theorem 8)**.** *Let $\mathcal{G}(N)$ denote a family of graphs on N vertices with space-complexity σ and time-complexity τ. Then a PRE scheme that is ciphertext-indistinguishable and weakly key-private for computationally bounded adversaries is also adaptively CPA-secure against computationally bounded adversaries for recoding graphs in $\mathcal{G}$ with a loss in security of $\approx \tau \cdot N^\sigma$. If the PRE is also statistically source-hiding then it is also adaptively HRA-secure.*

In many applications, the underlying recoding graph has a very particular structure like trees (or even paths) and low-depth graphs. For paths, or bounded-arity trees, our reduction only loses a quasi-polynomial factor. For low-depth graphs, the loss is exponential only in the depth (and thus polynomial for constant depth-graphs). Below, we mention two such applications.

1. In *key rotation* for encrypted cloud storage, a client has its data encrypted on a server, and occasionally wants to re-encrypt it (say, to restore security after key leakage). As the client does not trust the server, it will not want to hand it the decryption key. When using PRE, the client can simply send a re-key to the server, which enables it to locally re-encrypt all ciphertexts to the new key. In this application the recoding graph is simply a chain.

2. Another common application is *forwarding of encrypted email* without involving the receiver, say, for delegation during vacation or for filtering spam emails. In most cases the underlying delegation structure will be captured by simple graphs. For example, if delegation only happens to subordinates, the depth of the recoding graph is bounded by the depth of the hierarchy of the organisation.

Scheme	Setting	Assumption(s)	Hops
Construction 2 [AFGH05]	Bilinear maps	eDBDH and XDH	Single
Construction 4 [ABH09]	Bilinear maps	eDBDH and DLin	Single
Construction 5 [Gen09]	–	FHE	Multiple
Construction 7 [CCL$^+$14]	Lattices	LWE	Multiple

Table 4.1: PRE schemes we prove adaptively CPA and HRA secure (see Section 4.5 for the definitions of the assumptions).

Third Contribution: Adaptively-Secure PRE.

Finally, we show that the aforementioned three properties are satisfied by several existing constructions or by minor variants thereof, and thus Theorem 5 can be applied to them. An overview of these schemes is given in Table 4.1. We consider the most interesting corollary to our results the adaptive security of the LWE-based scheme by Chandran et al. [CCL$^+$14]:

Theorem 6 (Informal Theorem 13)**.** *The quasi-polynomially secure decisional LWE problem implies multi-hop, unidirectional adaptively CPA/HRA-secure PRE for chains or complete binary trees.*

4.2 Formal Definitions

4.2.1 Proxy Reencryption: Formal Definitions

Definition 16 (Multi-hop, unidirectional PRE)**.** A multi-hop, unidirectional PRE scheme for a message space $\mathcal{M}$ consists of the six-tuple of algorithms $(\mathsf{S}, \mathsf{K}, \mathsf{RK}, \mathsf{E}, \mathsf{D}, \mathsf{RE})$, which are explained below.

- $\mathsf{S}(1^\lambda, 1^L) \to \mathtt{pp}$: On input the security parameter λ and the maximum level L supported by the scheme (both in unary), **setup** outputs the public parameters $\mathtt{pp}$. We assume that $\mathtt{pp}$ is implicit in other function calls.
- $\mathsf{K}(\mathtt{pp}) \to (\mathtt{pk}, \mathtt{sk})$: **Key generation** returns a public key $\mathtt{pk}$ and the corresponding secret key $\mathtt{sk}$.
- $\mathsf{RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j) \to \mathtt{rk}_{i,j}$: On input a source key pair $(\mathtt{pk}_i, \mathtt{sk}_i)$ and a target public key $\mathtt{pk}_j$, **re-key generation** generates a unidirectional re-encryption key (rekey, for short) $\mathtt{rk}_{i,j}$.
- $\mathsf{E}(\mathtt{pk}, (\mathtt{m}, l)) \to (\mathtt{c}, l)$: **Encryption** takes as input the public key $\mathtt{pk}$, a message $\mathtt{m}$ and a level $l \in [1, L]$, and outputs a level-l ciphertext $(\mathtt{c}, l)$.
- $\mathsf{D}(\mathtt{sk}, (\mathtt{c}, l)) \to \mathtt{m}$: On input a ciphertext $(\mathtt{c}, l)$ and the secret key $\mathtt{sk}$, **decryption** outputs a message $\mathtt{m}$, or the symbol $\perp$ (if the ciphertext is invalid).
- $\mathsf{RE}(\mathtt{rk}_{i,j}, \mathtt{pk}_i, \mathtt{pk}_j, (\mathtt{c}_i, l)) \to (\mathtt{c}_j, l+1)$: **Reencryption** takes a re-key $\mathtt{rk}_{i,j}$, a source public key $\mathtt{pk}_i$, a target public key $\mathtt{pk}_j$ and a level-l ciphertext $\mathtt{c}_i$ under $\mathtt{pk}_i$ and transforms it to a level-$(l+1)$ ciphertext $\mathtt{c}_j$ under $\mathtt{pk}_j$. Only ciphertexts belonging to levels $l \in [1, L-1]$ can be re-encrypted. In constructions where arguments $\mathtt{pk}_i$ and/or $\mathtt{pk}_j$ are optional, we simply drop them.

Definition 16 differs slightly from the definition of multi-hop unidirectional PRE in [FL19]. Here, the re-keys are level-agnostic: the same re-key can be used to re-encrypt a ciphertext belonging to any level. In [FL19], however, a re-key associated to a level *cannot* be used to re-encrypt a ciphertext from a different level.

We require the PRE to satisfy the following two correctness properties.

Definition 17 (Correctness [ABH09])**.** A proxy re-encryption scheme (as in Definition 16) is correct w.r.t. the message space $\mathcal{M}$ if the following two properties hold:

1. *Correctness of encryption:* $\forall \lambda, L \in \mathbb{N},\ \forall\, \mathtt{pp} \in [\mathsf{S}(1^\lambda, 1^L)],\ \forall\, (\mathtt{pk}, \mathtt{sk}) \in [\mathsf{K}(\mathtt{pp})],$ $\forall\, (\mathtt{m}, l) \in \mathcal{M} \times [1, L]$:

$$\Pr\left[\mathsf{D}\left(\mathtt{sk}, \mathsf{E}(\mathtt{pk}, (\mathtt{m}, l))\right) \neq \mathtt{m}\right] = \mathsf{negl}(\lambda, L),$$

   where the probability is over the random coins of E.

2. *Correctness of re-encryption:* $\forall\, \lambda, L \in \mathbb{N},\ \forall\, \mathtt{pp} \in [\mathsf{S}(1^\lambda, 1^L)],\ \forall\, (\mathtt{pk}_i, \mathtt{sk}_i),\ (\mathtt{pk}_j, \mathtt{sk}_j)$ $\in [\mathsf{K}(\mathtt{pp})],\ \forall\, \mathtt{rk}_{i,j} \in [\mathsf{RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)],\ \forall\, (\mathtt{m}, l) \in \mathcal{M} \times [1, L-1]$:

$$\Pr\left[\mathsf{D}\left(\mathtt{sk}_j, \mathsf{RE}(\mathtt{rk}_{i,j}, \mathtt{pk}_i, \mathtt{pk}_j, (\mathtt{c}_i, l))\right) \neq \mathtt{m}\right] = \mathsf{negl}(\lambda, L),$$

   where $(\mathtt{c}_i, l)$ is a level-l ciphertext of $\mathtt{m}$ under $\mathtt{pk}_i$ resulting either from direct encryption or reencryption of a level-$(l-1)$ ciphertext, and the probability is over the random coins of E and RE.

4.2.2 Modelling Security

Selective Corruption.

The selective security of a multi-hop, unidirectional PRE scheme against a chosen-plaintext attack is modelled using the security game given in Algorithm 4.1.[1] It is an extension of the security model for single-hop PRE from [ABH09] to the multi-hop setting.[2] The limiting feature of the model is that the adversary has to fix, beforehand in Phase 1, the honest and corrupt public keys. Its goal is to distinguish an encryption of $\mathtt{m}_0$ from an encryption of $\mathtt{m}_1$ (for $\mathtt{m}_0, \mathtt{m}_1$ of its choice) under a key of its choice. The game aborts if the adversary does one of the following:

- query the challenge oracle on a corrupt public key ($\mathtt{abort}_1$);
- request a re-key from an honest key to a corrupt key ($\mathtt{abort}_2$);
- query a re-encryption from an honest to a corrupt key ($\mathtt{abort}_3$).

Definition 18 (sPRE-CPA-security)**.** A PRE scheme is (t, ϵ)-selectively secure against chosen-plaintext attack if $\mathsf{sCPA}^0 \approx_{(t,\epsilon)} \mathsf{sCPA}^1$, where sCPA^b is defined in Algorithm 4.1.

[1] The formulation here is slightly different from the original one in [ABH09]. In [ABH09], the adversary has access to two oracles in Phase 1, one for generating honest keys (i.e., the adversary gets just the public key) and the other for generating corrupted keys (i.e., the adversary gets both public and secret key). In Algorithm 4.1 the adversary is first given all the public keys and can then, in Phase 1, choose the keys it wants to corrupt.

[2] [FL19] formalised security differently; we stick to the definition from [ABH09].

Algorithm 4.1: sPRE-CPA security game

Challenger $\mathsf{sCPA}^b(1^\lambda, 1^L, N)$

Set $\mathcal{C} = \emptyset$ // Stores the corrupt public keys

$\mathtt{pp} \leftarrow \mathsf{PRE.S}(1^\lambda, 1^L)$, $(\mathtt{pk}_1, \mathtt{sk}_1), \ldots, (\mathtt{pk}_N, \mathtt{sk}_N) \leftarrow \mathsf{PRE.K}(\mathtt{pp})$ // Generate keys

$\forall i, j \in [1, N], i \neq j : \mathtt{rk}_{i,j} \leftarrow \mathsf{PRE.RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$ // Generate re-keys

$\mathsf{state} \leftarrow \mathsf{A}_1^{(\mathtt{corrupt},\cdot)}(\mathtt{pp})$ // Phase 1

$b' \leftarrow \mathsf{A}_2^{(\mathtt{rekey},\cdot,\cdot),(\mathtt{reencrypt},\cdot,\cdot,\cdot),(\mathtt{challenge},\cdot,\cdot,\cdot)}(\mathtt{pk}_1, \ldots, \mathtt{pk}_N, \mathsf{state})$ // Phase 2

return b'

Oracle $(\mathtt{corrupt}, i)$

Add i to $\mathcal{C}$

return $\mathtt{sk}_i$

Oracle $(\mathtt{rekey}, i, j)$

if $i \notin \mathcal{C}$ *and* $j \in \mathcal{C}$ **then**

  HALT // $\mathtt{abort}_2$

return $\mathtt{rk}_{i,j}$

Oracle $(\mathtt{reencrypt}, i, j, (\mathtt{c}_i, l))$

if $i \notin \mathcal{C}$ *and* $j \in \mathcal{C}$ **then**

  HALT // $\mathtt{abort}_3$

return $(\mathtt{c}_j, l+1) \leftarrow \mathsf{PRE.RE}(\mathtt{rk}_{i,j}, \mathtt{pk}_i, \mathtt{pk}_j, (\mathtt{c}_i, l))$

Oracle $(\mathtt{challenge}, i^*, (\mathtt{m}_0^*, \mathtt{m}_1^*), l^*)$ // Single access

if $i^* \in \mathcal{C}$ **then**

  HALT // $\mathtt{abort}_1$

return $(\mathtt{c}_{i^*}, l^*) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_{i^*}, (\mathtt{m}_b^*, l^*))$

Security against honest-reencryption attack. A stronger security definition was introduced in [Coh19] to address some of the restrictions that sPRE-CPA imposes on the adversary. The idea is to allow re-encryptions from honest to corrupt keys, if the ciphertexts to re-encrypt were honestly generated. The adversary can obtain such honest ciphertexts via an `encrypt` oracle, which stores them in a list. The `reencrypt` oracle now takes the index of an honestly generated ciphertext. It was shown in [Coh19] that (selective) HRA-security implies (selective) CPA-security and also that if the PRE scheme is re-encryption-simulatable (a generalization of Definition 24) then (selective) CPA-security implies (selective) HRA-security. In sPRE-HRA, which we formally define in Algorithm 4.2, $\mathtt{abort}_3$ is relaxed to

- $\mathtt{abort}_3^*$: The adversary queries the re-encryption of a ciphertext that is the result of a chain of re-encryptions of the challenge ciphertext from an honest to a corrupt key.

Definition 19 (sPRE-HRA-security)**.** A PRE scheme is (t, ϵ)-selectively secure against honest-reencryption attack if $\mathsf{sHRA}^0 \approx_{(t,\epsilon)} \mathsf{sHRA}^1$, where sHRA^b is defined in Algorithm 4.2.

Algorithm 4.2: sPRE-HRA security game

Challenger $\mathsf{sHRA}^b(1^\lambda, 1^L, N)$

Set $\mathcal{C}, \mathcal{E} = \emptyset$ // $\mathcal{C}$ stores corrupt keys, $\mathcal{E}$ re-keys and re-encryptions

Set $ctr = 0$ // Counts ciphertexts generated

Set $\mathcal{L}, \mathcal{L}^* = \emptyset$ // Stores honest ciphertexts and which derived from challenge

$\mathtt{pp} \leftarrow \mathsf{PRE.S}(1^\lambda, 1^L)$, $(\mathtt{pk}_1, \mathtt{sk}_1), \ldots, (\mathtt{pk}_N, \mathtt{sk}_N) \leftarrow \mathsf{PRE.K}(\mathtt{pp})$ // Generate keys

$\forall i, j \in [1, N], i \neq j : \mathtt{rk}_{i,j} \leftarrow \mathsf{PRE.RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$ // Generate re-keys

$\mathsf{state} \leftarrow \mathsf{A}_1^{(\mathtt{corrupt},\cdot)}(\mathtt{pp})$ // Phase 1

$b' \leftarrow \mathsf{A}_2^{(\mathtt{encrypt},\cdot,\cdot),(\mathtt{rekey},\cdot,\cdot),(\mathtt{reencrypt},\cdot,\cdot,\cdot),(\mathtt{challenge},\cdot,\cdot,\cdot)}(\mathtt{pk}_1, \ldots, \mathtt{pk}_N, \mathsf{state})$ // Phase 2

return b'

Oracles `corrupt` and `rekey` are defined like in Algorithm 4.1.

Oracle $(\mathtt{encrypt}, i, (\mathtt{m}, l))$

$(\mathtt{c}, l) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_i, (\mathtt{m}, l))$

Increment *ctr* and add $(ctr, i, \mathtt{m}, (\mathtt{c}, l))$ to $\mathcal{L}$

return $(\mathtt{c}, l)$

Oracle $(\mathtt{reencrypt}, i, j, k)$

Retrieve $(k, i, \mathtt{m}, (\mathtt{c}_i, l))$ from $\mathcal{L}$ and increment *ctr*

$(\mathtt{c}_j, l+1) \leftarrow \mathsf{PRE.RE}(\mathtt{rk}_{i,j}, \mathtt{pk}_i, \mathtt{pk}_j, (\mathtt{c}_i, l))$

if $k \in \mathcal{L}^*$ **then** // The ciphertext is derived from the challenge

  if $j \in \mathcal{C}$ **then** HALT

  else add *ctr* to $\mathcal{L}^*$ // $\mathtt{abort}_3^*$

Add $(ctr, j, \mathtt{m}, (\mathtt{c}_j, l+1))$ to $\mathcal{L}$

return $(\mathtt{c}_j, l+1)$

Oracle $(\mathtt{challenge}, i^*, (\mathtt{m}_0^*, \mathtt{m}_1^*), l^*)$ // Single access

Compute $(\mathtt{c}_{i^*}, l^*) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_{i^*}, (\mathtt{m}_b^*, l^*))$ and increment *ctr*

if $i^* \in \mathcal{C}$ **then** HALT

else add *ctr* to $\mathcal{L}^*$ // $\mathtt{abort}_1$

Add $(ctr, i^*, \mathtt{m}_b^*, (\mathtt{c}_{i^*}, l^*))$ to $\mathcal{L}$

return $(\mathtt{c}_{i^*}, l^*)$

Modelling Adaptive Corruption.

The adaptive security games corresponding to Algorithms 4.1 and 4.2 are given in Algorithms 4.3 and 4.4, respectively. To model adaptive corruption, we think of the game being played on a directed graph $G = (\mathcal{V}, \mathcal{E})$ called the "recoding" graph. The vertices of the recoding graph correspond to the public keys, i.e., $\mathcal{V} = [1, N]$. The edges are derived from the re-keys and re-encryptions issued to the adversary in the security game, and their purpose is to ensure that the adversary does not win the game in a trivial manner. In particular, the recoding graph is defined so that *no* corrupt key is reachable from the challenge key. To be precise, in CPA an

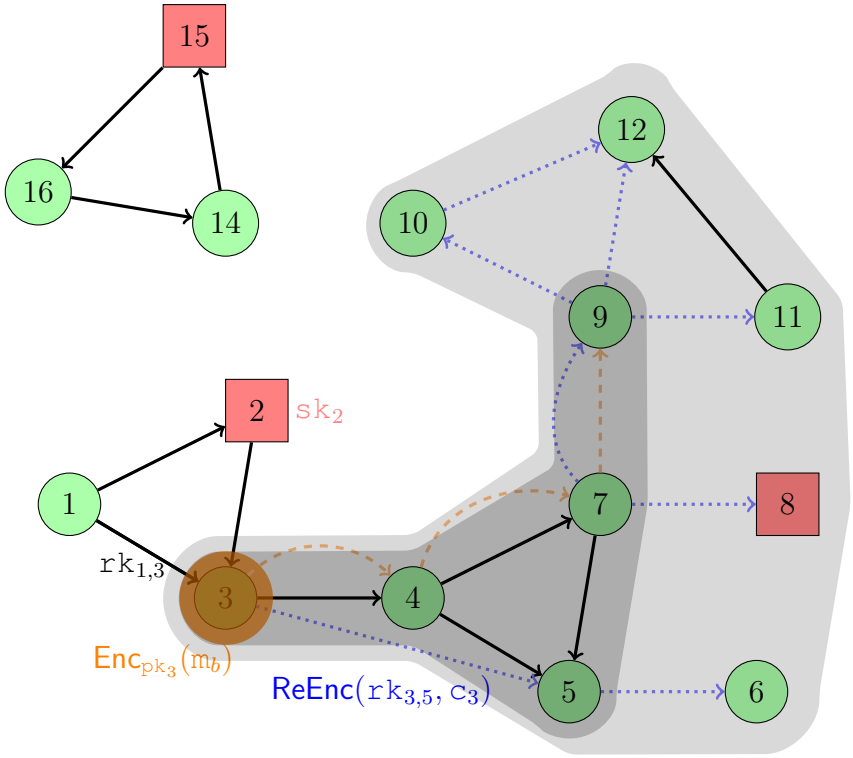


Figure 4.1: Recoding graph. The round green nodes represent the honest users, whereas the square red nodes are the corrupted users. The edges denote the recoding information. In particular, a solid black edge from node i to node j represents that a rekey $\mathtt{rk}_{i,j}$ from i to j was issued. Similarly, the dashed orange edges represent the re-encryptions related to the challenge ciphertext (therefore, 3 is the challenge vertex) and dotted blue edges represent the remaining re-encryptions. For CPA, all the edges are counted, but for HRA the dotted blue edges are *not* counted. The subgraph of the recoding graph that forms the challenge graph (cf. Section S:AS) is shaded: the darker inner shading for HRA, whereas the lighter outer shading is the challenge graph for CPA. Note that the edge $(7, 8)$ is valid in the case of HRA, but invalid for CPA (and therefore the CPA challenger would abort at the end of such an execution.)

edge (i, j) is added to $\mathcal{E}$ if the adversary made either a $(\mathtt{rekey}, i, j)$ or $(\mathtt{reencrypt}, i, j, \cdot)$ query (see Algorithm 4.3 and Figure 4.1). Consequently, the adversary is forbidden from making *any* re-key or re-encryption queries to a corrupt user that is reachable from the challenge key.[3]

For HRA, on the other hand, (i, j) is added to $\mathcal{E}$ if the adversary made either a $(\mathtt{rekey}, i, j)$ query or a $(\mathtt{reencrypt}, i, j, k)$ query where the k-th ciphertext is a re-encryption of the challenge ciphertext (see Algorithm 4.4 and Figure 4.1). This is less restrictive than in CPA: the adversary can make re-encryption queries to a corrupt user that is reachable from the challenge key *unless* the ciphertext is related to the challenge ciphertext.

Definition 20 (PRE-CPA-security)**.** A PRE scheme is (t, ϵ)-adaptively secure against chosen-plaintext attack if $\mathrm{CPA}^0 \approx_{(t,\epsilon)} \mathrm{CPA}^1$, where CPA^b is defined in Algorithm 4.3.

Definition 21 (PRE-HRA-security)**.** A PRE scheme is (t, ϵ)-adaptively secure against honest-reencryption attack if $\mathrm{HRA}^0 \approx_{(t,\epsilon)} \mathrm{HRA}^1$, where HRA^b is defined in Algorithm 4.4.

[3]The selective CPA notion (Algorithm 4.1) is in fact more restrictive in that it does not allow re-keys and re-encryptions from *any* honest user to a corrupt user.

Algorithm 4.3: PRE-CPA security game

Challenger $\mathsf{CPA}^b(1^\lambda, 1^L, N)$

1 Set $\mathcal{C}, \mathcal{E} = \emptyset$ `// C stores corrupt keys, E re-keys and re-encryptions`

2 $\mathtt{pp} \leftarrow \mathsf{PRE.S}(1^\lambda, 1^L)$, $(\mathtt{pk}_1, \mathtt{sk}_1), \ldots, (\mathtt{pk}_N, \mathtt{sk}_N) \leftarrow \mathsf{PRE.K}(\mathtt{pp})$ `// Generate keys`

3 $\forall i, j \in [1, N], i \neq j : \mathtt{rk}_{i,j} \leftarrow \mathsf{PRE.RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$ `// Generate re-keys`

4 $b' \leftarrow \mathsf{A}^{(\mathtt{corrupt},\cdot),(\mathtt{rekey},\cdot,\cdot),(\mathtt{reencrypt},\cdot,\cdot,\cdot),(\mathtt{challenge},\cdot,\cdot,\cdot)}(\mathtt{pp}, \mathtt{pk}_1, \ldots, \mathtt{pk}_N)$

5 **if** A *made call* $(\mathtt{challenge}, i^*, \cdot, \cdot)$ *for some* i^* `// Check abort conditions`

6 **then**

7 | **if** $\exists i \in \mathcal{C} : i^*$ *is connected to* i *in* $([1, N], \mathcal{E})$ **then**

8 | | **return** 0

9 **return** b'

Oracle $(\mathtt{corrupt}, i)$

10 Add i to $\mathcal{C}$

11 **return** $\mathtt{sk}_i$

Oracle $(\mathtt{rekey}, i, j)$

12 Add (i, j) to $\mathcal{E}$ `// Add to recoding graph`

13 **return** $\mathtt{rk}_{i,j}$

Oracle $(\mathtt{reencrypt}, i, j, (\mathtt{c}_i, l))$

14 Add (i, j) to $\mathcal{E}$ `// Add to recoding graph`

15 **return** $(\mathtt{c}_j, l+1) \leftarrow \mathsf{PRE.RE}(\mathtt{rk}_{i,j}, \mathtt{pk}_i, \mathtt{pk}_j, (\mathtt{c}_i, l))$

Oracle $(\mathtt{challenge}, i^*, (\mathtt{m}_0^*, \mathtt{m}_1^*), l^*)$ `// Single access`

16 **return** $(\mathtt{c}_{i^*}, l^*) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_{i^*}, (\mathtt{m}_b^*, l^*))$

4.3 Preliminaries

This section provides the background necessary for the main results in Section 4.4. We start with the security assumptions on PRE that allow us to prove adaptive security (Section 4.3.1) and then give the description of the pebbling game that is used in the design of the hybrids (Section 4.3.2).

4.3.1 Security Assumptions on PRE

In this section we describe the three security properties of PRE schemes that allow us to prove adaptive security: indistinguishability, key-privacy and source-hiding.

Indistinguishability of ciphertexts. For proxy re-encryption, we require the notion of indistinguishability, as defined for public-key encryption in [GM82], to hold on *all* levels:

Definition 22 (Indistinguishability)**.** A proxy re-encryption scheme PRE has (t, ϵ)-indistinguishable ciphertexts if $\mathsf{IND}^0 \approx_{(t,\epsilon)} \mathsf{IND}^1$ with IND as in Algorithm 4.5.

Algorithm 4.4: PRE-HRA security game

Challenger $\mathsf{HRA}^b(1^\lambda, 1^L, N)$

Set $\mathcal{C}, \mathcal{L}, \mathcal{L}^* = \emptyset$ and $ctr = 0$ // $\mathcal{L}$ stores honest enc's, $\mathcal{L}^*$ marks challenge reenc's

$\mathcal{E} = \emptyset$ // The edges of the recoding graph

$\mathtt{pp} \leftarrow \mathsf{PRE.S}(1^\lambda, 1^L)$, $(\mathtt{pk}_1, \mathtt{sk}_1), \ldots, (\mathtt{pk}_N, \mathtt{sk}_N) \leftarrow \mathsf{PRE.K}(\mathtt{pp})$ // Generate keys

$\forall i, j \in [1, N], i \neq j : \mathtt{rk}_{i,j} \leftarrow \mathsf{PRE.RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$ // Generate re-keys

$b' \leftarrow \mathsf{A}^{(\mathtt{corrupt},\cdot),(\mathtt{rekey},\cdot,\cdot),(\mathtt{encrypt},\cdot,\cdot),(\mathtt{reencrypt},\cdot,\cdot,\cdot),(\mathtt{challenge},\cdot,\cdot,\cdot)}(\mathtt{pp}, \mathtt{pk}_1, \ldots, \mathtt{pk}_N)$

if A *made call* $(\mathtt{challenge}, i^*, \cdot, \cdot)$ *for some* i^* // Check abort conditions

then

  if $\exists i \in \mathcal{C} : i^*$ *is connected to* i **then**

    return 0

return b'

Oracle $(\mathtt{corrupt}, i)$

Add i to $\mathcal{C}$

return $\mathtt{sk}_i$

Oracle $(\mathtt{rekey}, i, j)$

Add (i, j) to $\mathcal{E}$ // Add to recoding graph

return $\mathtt{rk}_{i,j}$

Oracle $(\mathtt{encrypt}, i, (\mathtt{m}, l))$

$c \leftarrow \mathsf{PRE.E}(\mathtt{pk}_i, (\mathtt{m}, l))$, increment *ctr* and add $(ctr, i, \mathtt{m}, (\mathtt{c}, l))$ to $\mathcal{L}$

return c

Oracle $(\mathtt{reencrypt}, i, j, k)$

Retrieve $(k, i, \mathtt{m}, (\mathtt{c}_i, l))$ from $\mathcal{L}$

$(\mathtt{c}_j, l + 1) \leftarrow \mathsf{PRE.RE}(\mathtt{rk}_{i,j}, \mathtt{pk}_i, \mathtt{pk}_j, (\mathtt{c}_i, l))$

Increment *ctr* and add $(ctr, j, \mathtt{m}, (\mathtt{c}_j, l + 1))$ to $\mathcal{L}$

if $k \in \mathcal{L}^*$ **then** // $\mathtt{c}_j$ derived from challenge

  Add *ctr* to $\mathcal{L}^*$ and add (i, j) to $\mathcal{E}$ // Add to recoding graph

return $(\mathtt{c}_j, l + 1)$

Oracle $(\mathtt{challenge}, i^*, (\mathtt{m}_0^*, \mathtt{m}_1^*), l^*)$ // Single access

Compute $(\mathtt{c}_{i^*}, l^*) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_{i^*}, (\mathtt{m}_b^*, l^*))$

Increment *ctr*, add $(ctr, i^*, \mathtt{m}_b^*, (\mathtt{c}_{i^*}, l^*))$ to $\mathcal{L}$ and *ctr* to $\mathcal{L}^*$

return $(\mathtt{c}_{i^*}, l^*)$

Key-privacy. The original notion of key-privacy for PRE, which we refer to as "strong" key-privacy, was introduced in [ABH09]. It is modelled by a security game similar to sPRE-CPA: the adversary has access to `corrupt`, `rekey` and `reencrypt` oracles, but as a challenge it has to distinguish a real re-key from a re-key sampled *uniformly at random* from the support of re-keys. We refer the readers to [ABH09] for the details.

Algorithm 4.5: Security game IND for ciphertext indistinguishability

Challenger $\mathsf{IND}^b(1^\lambda, 1^L)$
1 $\mathtt{pp} \leftarrow \mathsf{PRE.S}(1^\lambda, 1^L)$, $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{PRE.K}(\mathtt{pp})$
2 **return** $b' \leftarrow \mathsf{A}^{(\mathtt{challenge},\cdot,\cdot)}(\mathtt{pp}, \mathtt{pk})$

Oracle $(\mathtt{challenge}, (\mathtt{m}_0^*, \mathtt{m}_1^*), l^*)$
3 **return** $\mathsf{PRE.E}(\mathtt{pk}, (\mathtt{m}_b^*, l^*))$

Algorithm 4.6: Security game KP for weak key-privacy

Challenger $\mathsf{KP}^b(1^\lambda, 1^L)$
1 $\mathtt{pp} \leftarrow \mathsf{PRE.S}(1^\lambda, 1^L)$, $(\mathtt{pk}_0, \mathtt{sk}_0), \ldots, (\mathtt{pk}_{\delta_{out}}, \mathtt{sk}_{\delta_{out}}) \leftarrow \mathsf{K}(\mathtt{pp})$
2 $\forall j \in [1, \delta_{out}] : \mathtt{rk}_{0,j}^{(0)} \leftarrow \mathsf{RK}((\mathtt{pk}_0, \mathtt{sk}_0), \mathtt{pk}_j)$
3 $\mathtt{rk}_{0,j}^{(1)} \leftarrow \mathsf{RK}^*(\mathtt{pp}, \mathtt{pk}_j)$
4 **return** $b' \leftarrow \mathsf{A}(\mathtt{pp}, \mathtt{pk}_0, \ldots, \mathtt{pk}_{\delta_{out}}, \mathtt{rk}_{0,1}^{(b)}, \ldots, \mathtt{rk}_{0,\delta_{out}}^{(b)})$

We only need a weaker definition stating that re-keys should hide the source keys. That is, the re-key $\mathtt{rk}_{0,1}$ from source $(\mathtt{pk}_0, \mathtt{sk}_0)$ to a target key $\mathtt{pk}_1$ should be indistinguishable from a random source to $\mathtt{pk}_1$. In addition, we need this property to hold with respect to multiple re-keys. More formally, the security game for weak key-privacy is given in Algorithm 4.6 where the simulator RK^* is defined as

$$\mathsf{RK}^*(\mathtt{pp}, \mathtt{pk}_1) := \mathsf{RK}((\mathtt{pk}_0, \mathtt{sk}_0), \mathtt{pk}_1) : (\mathtt{pk}_0, \mathtt{sk}_0) \leftarrow \mathsf{K}(\mathtt{pp}).$$

Definition 23 (Weak key-privacy)**.** Let $\delta_{out} \in \mathbb{N}$. A proxy re-encryption scheme PRE is $(t, \epsilon, \delta_{out})$-weakly key-private if $\mathsf{KP}^0 \approx_{(t,\epsilon)} \mathsf{KP}^1$ with KP as in Algorithm 4.6.

Source-hiding. Source-hiding is a special case of re-encryption-simulatability, a notion that was introduced in [Coh19]. It requires that re-encryptions can be simulated without knowledge of the secret key. In particular, the simulated re-encryptions should be indistinguishable from re-encrypted ciphertexts even when given the secret keys for the source and target public keys, as well as the re-key that was used for re-encryption (hence the notion of indistinguishability is at least that of statistical indistinguishability). A PRE scheme is called *source-hiding* if re-encrypted ciphertexts have the same distribution as "fresh" ciphertexts, i.e., the encryption algorithm can be used as a simulator for re-encryption.

Definition 24 (Source-hiding)**.** A proxy re-encryption scheme PRE is (t, ϵ)-source-hiding if $\mathsf{SH}^0 \approx_{(t,\epsilon)} \mathsf{SH}^1$, with SH as defined in Algorithm 4.7.

4.3.2 Pebbling Games

In Chapter 3 we already saw how pebbling games on graphs can be used to define suitable hybrid games; see Section 3.3.2, Definition 8. While for GSD we required an *edge*-pebbling game, in the setting of PRE we consider a variant of the original reversible *node*-pebbling game from [Ben89]. In particular, the rules are exactly the opposite of those in [Ben89] (see

Algorithm 4.7: Security game SH for source hiding

Challenger $\mathsf{SH}^b(1^\lambda, 1^L)$
1 $\mathtt{pp} \leftarrow \mathsf{PRE.S}(1^\lambda, 1^L)$
2 $(\mathtt{pk}_0, \mathtt{sk}_0), (\mathtt{pk}_1, \mathtt{sk}_1) \leftarrow \mathsf{PRE.K}(\mathtt{pp})$
3 $\mathtt{rk}_{0,1} \leftarrow \mathsf{PRE.RK}((\mathtt{pk}_0, \mathtt{sk}_0), \mathtt{pk}_1)$
4 $b' \leftarrow \mathsf{A}^{(\mathtt{challenge},\cdot,\cdot)}(\mathtt{pp}, (\mathtt{pk}_0, \mathtt{sk}_0), (\mathtt{pk}_1, \mathtt{sk}_1), \mathtt{rk}_{0,1})$
5 **return** b'

Oracle $(\mathtt{challenge}, \mathtt{m}^*, l^*)$ // $l^* \in [1, L-1]$
6 $(\mathtt{c}_0, l^*) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_0, (\mathtt{m}^*, l^*))$
7 $(\mathtt{c}_1^{(0)}, l^*+1) \leftarrow \mathsf{PRE.RE}(\mathtt{rk}_{0,1}, \mathtt{pk}_0, \mathtt{pk}_1, (\mathtt{c}_0, l^*))$ // Real re-encryption
8 $(\mathtt{c}_1^{(1)}, l^*+1) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_1, (\mathtt{m}^*, l^*+1))$ // Simulate re-encryption
9 **return** $(\mathtt{c}_0, \mathtt{c}_1^{(b)})$

Definition 61 in Chapter 7): a pebble can be placed on or removed from a vertex if all its *children* carry a pebble, and the goal is to place a pebble on some target *sources*.[4]

Definition 25. A *reversible source-pebbling* of a directed acyclic graph $G = (\mathcal{V}, \mathcal{E})$ with a unique source vertex i^* is a sequence $\boldsymbol{\mathcal{P}} := (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ of pebbling configurations $\mathcal{P}_\ell \subseteq \mathcal{V}$. Two subsequent configurations differ only in one vertex and the following rule is respected in a move: a pebble can be placed on or removed from a vertex iff all its *children* carry a pebble. That is, $\boldsymbol{\mathcal{P}}$ is a valid sequence iff

$$\forall \ell \in [1, \tau]\ \exists! \, i \in \mathcal{P}_{\ell-1} \triangle \mathcal{P}_\ell \ \text{ and } \ \mathsf{children}(i, G) \subseteq \mathcal{P}_{\ell-1}.$$

Starting with an empty graph (i.e., $\mathcal{P}_0 = \emptyset$), the goal of the game is to place a pebble on the source (i.e., $i^* \in \mathcal{P}_\tau$).

For a DAG G, let $\mathcal{P}_G$ denote the set of all valid reversible source-pebbling sequences (as per Definition 25) for G. The *time complexity* of a *particular* sequence $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ for a DAG G is defined as $\tau_{\boldsymbol{\mathcal{P}}}(G) := \tau$, whereas its space complexity is defined as

$$\sigma_{\boldsymbol{\mathcal{P}}}(G) := \max_{\ell \in [0,\tau]} |\mathcal{P}_\ell|.$$

Definition 26 (Space- and time-complexity of a class of DAGs)**.** We say that a class of DAGs $\mathcal{G}$ has time complexity τ and space complexity σ if

$$\forall G \in \mathcal{G}\ \exists \boldsymbol{\mathcal{P}} \in \mathcal{P}_G : \tau_{\boldsymbol{\mathcal{P}}}(G) \leq \tau \wedge \sigma_{\boldsymbol{\mathcal{P}}}(G) \leq \sigma.$$

Concrete Bounds. We compute upper bounds on the reversible source-pebbling complexity for the following classes of *single-source* graphs on N vertices:

[Lemma 7]: $\mathcal{G}(N, \delta_{out}, D)$, DAGs of size N with outdegree δ_{out} and depth D;
[Lemma 8]: $\mathcal{B}(N) \subset \mathcal{G}(N, 2, \log N)$, complete binary trees of size N; and
[Lemma 9]: $\mathcal{C}(N) \subset \mathcal{G}(N, 1, N)$, chains of length N.

[4] Alternatively, one can think of the pebbling game in Definition 25 as the classical reversible pebbling game played on a DAG whose edges have their direction flipped.

Algorithm 4.8: A pebbling strategy for general DAGs.

```
P_1(G, ℓ)
Set T = 0 and P = ∅                                   // Global variables
Let i* denote the source of the graph G
return P'_1(G, ℓ, i*, i*)

P'_1(G, T, i*, i)          // i* denotes the source; i is the vertex currently
 pebbled
for j ∈ children(i, G) do
    P'_1(G, T, i*, j)                                 // Pebble children recursively
if i ∈ P then
    P := P \ {i}                                      // Unpebble if i already pebbled
else
    P := P ∪ {i}                                      // Place pebble on i
    if i = i* then
        return P                                      // Placed pebble on source
Increment T
if T = ℓ then
    return P          // P currently stores the ℓth pebbling configuration
for j ∈ children(i, G) do
    P'_1(G, T, i*, j)                                 // Unpebble children
```

Note that the same bounds hold for normal reversible node-pebbling on the respective classes of graphs that are obtained by flipping the direction of the edges. The algorithms for arbitrary DAGs and chains are closely related to those presented in Section 3.3.2 for reversible edge-pebbling.[5]

Lemma 7 (Arbitrary DAGs)**.** $\mathcal{G}(N, \delta_{out}, D)$ *has space-complexity* $(\delta_{out}+1)\cdot D$ *and time-complexity* $(2\delta_{out})^D$.

Proof. The pebbling algorithm P_1 that pebbles any graph in $\mathcal{G}(N, \delta_{out}, D)$ using at most $(\delta_{out}+1)\cdot D$ pebbles in at most $(2\delta_{out})^D$ moves is given in Algorithm 4.8. The strategy is recursive in the depth, and to pebble a vertex i, P_1 (recursively) pebbles all of i's children. We consider the time and space complexity of the pebbling sequence defined by P_1 as functions $\tau(D)$ and $\sigma(D)$ of the depth D. Then the number of moves incurred is captured by the expression $\tau(D) \leq 2\delta_{out}\cdot\tau(D-1)$ with $\tau(1) \leq 2\delta_{out}$, and hence $\tau(\delta_{out}) \leq (2\delta_{out})^D$. The number of pebbles, on the other hand, is captured by the recursion $\sigma(D) < (\delta_{out}+1)+\sigma(D-1)$ with $\sigma(1) = \delta_{out}+1$; hence $\sigma(D) = (\delta_{out}+1)\cdot D$. □

Lemma 8 (Complete binary trees)**.** $\mathcal{B}(N)$ *has space-complexity* $3\cdot\log N$ *and time-complexity* N^2.

Proof. This follows from Lemma 7 on substituting $\delta_{out} = 2$ and $D = \log N$. □

Lemma 9 (Chains)**.** $\mathcal{C}(N)$ *has space-complexity* $\log N + 1$ *and time-complexity* $3^{\log N}$.

[5] Actually, one can view node-pebbling as a special case of edge-pebbling; see Section 7.3.1 in Chapter 7.

Algorithm 4.9: A pebbling strategy for chains with N vertices, for N a power of two.

```
   P_2(N, ℓ)                                  // N denotes the length of the chain
 1 Set T = 0 and P = ∅                                    // Global variables
 2 return P'_2(1, N, T)

   P'_2(i, j, T)          // i and j denote the end points of the active chain
 3 if i = j then                                          // End of recursion
 4 |  if i ∈ P then
 5 |  |  P := P \ {i}                          // Unpebble if i already pebbled
 6 |  else
 7 |  |  P := P ∪ {i}                                    // Place pebble on i
 8 |  |  if i = 1 then                        // Placed pebble on challenge
 9 |  |  |  return P
10 |  Increment T
11 |  if T = ℓ then
12 |  |  return P     // P currently stores the ℓth pebbling configuration
13 else
14 |  P'_2(i, (i + j − 1)/2, T)                  // Recursively pebble left half
15 |  P'_2((i + j + 1)/2, j, T)                 // Recursively pebble right half
16 |  P'_2(i, (i + j − 1)/2, T)                 // Recursively unpebble left half
```

Proof. A pebbling algorithm P_2 for pebbling the source vertex of $\mathcal{C}(N)$, where N is a power of two, is given in Algorithm 4.9 – the argument is similar to the case of edge pebbling (see Section 3.3.2) and can easily be extended for arbitrary N and vertex. Let $\sigma(N)$ and $\tau(N)$ denote the space and time complexity of the pebbling defined by P_2 for chains of length N. Then the number of pebbles used by P_2 is captured by the recursion $\sigma(N) = \sigma(N/2) + 1$, with $\sigma(1) = 1$. The number of moves, on the other hand, is captured by $\tau(N) = 3 \cdot \tau(N/2)$ with $\tau(1) = 1$. Therefore, $\sigma(N) = \log N + 1$ and $\tau(N) = 3^{\log N}$. □

4.4 Framework for Adaptive Security

In this section we demonstrate, using the Piecewise-Guessing framework from Chapter 3, how adaptive security can be achieved for PRE. In particular, we show that for CPA and derive an analogous result for HRA. Similar to the applications given in Chapter 3, we use pebbling games on DAGs to design the hybrid games. Each pebbling configuration uniquely determines a hybrid game bridging the two real games CPA^0 and CPA^1. The DAG that we pebble in the proof is the subgraph of the recoding graph that is reachable from the challenge i^* (via the edges $\mathcal{E}$ defined during the game); it is thus a subgraph of the recoding graph with one unique source i^*, which we call the *challenge graph*. A pebble on a vertex allows the simulation of the hybrid to be carried out *without* the knowledge of the secret key associated with that vertex. The pebbling rules will ensure that hybrids corresponding to two successive pebbling configurations can be proven indistinguishable assuming key-privacy.

4.4.1 Adaptive Security Against Chosen-Plaintext Attack

We first show how a pebbling sequence on the challenge graph defines a sequence of fully selective hybrids (Lemma 10), and then prove that these hybrids are partially selectivised (Lemma 11).

Fully Selective Hybrids.

In the fully selectivised version of PRE-CPA (Algorithm 4.3), A first makes a commitment $\hat{G}$ to the challenge graph. Any correct commitment $\hat{G}$ must therefore have one unique source, which we denote by $\hat{\imath}$. The selective challenger is thus $\mathrm{SEL}_{\mathcal{G}}[\mathsf{CPA}^b, \mathsf{w}]$, where w is the function that extracts the recoding graph G and the challenge user i^* from the transcript and returns the challenge graph, i.e., the subgraph of G reachable from i^*. Note that this is fundamentally different from the original selective game (i.e., sCPA in Algorithm 4.1) where the adversary commits, beforehand, to *the set of corrupt public keys*.

Each hybrid is associated with a pebbling configuration $\mathcal{P}_\ell$ and a bit b, and we consider the sequence of hybrids $\mathsf{H}^0_0, \ldots, \mathsf{H}^0_\tau, \mathsf{H}^1_\tau, \ldots, \mathsf{H}^1_0$. The pebbling state of a vertex dictates how the outgoing re-key and re-encrypt queries are simulated, whereas the bit determines how the challenge query is answered. To be precise, in game H^b_ℓ, for each pebbled vertex in $\mathcal{P}_\ell$ all used re-keys outgoing from that vertex are faked, and the challenge query is answered by an encryption of m^*_b. (Rekeys outgoing from pebbled vertices that are not used for any queries are defined as real re-keys.) Observe that the secret key corresponding to a vertex is used only for the generation of the re-keys outgoing from that vertex; the simulation of a hybrid can thus be carried out *without* knowledge of the secret keys corresponding to the pebbled vertices (as the non-queried re-keys need not be generated).

Since the initial pebbling configuration is the empty set, H^0_0 and H^1_0 correspond to the (fully selectivised) games $\mathrm{SEL}_{\mathcal{G}}[\mathsf{CPA}^0, \mathsf{w}]$ and $\mathrm{SEL}_{\mathcal{G}}[\mathsf{CPA}^1, \mathsf{w}]$, respectively. Now, consider the middle hybrids H^0_τ and H^1_τ: they are the same except for the response to the challenge query which is the encryption of $\mathtt{m}^*_0$ in the former and the encryption of $\mathtt{m}^*_1$ in the latter. Since the pebbling configuration $\mathcal{P}_\tau$, by definition, contains a pebble on the challenge vertex i^*, the simulation of this hybrid can be carried out without knowledge of the secret key corresponding to i^*. This means we can prove the indistinguishability of these two hybrids from the indistinguishability of the PRE scheme. To be precise, the reduction embeds the challenge public key at $\hat{\imath}$, which is defined by the commitment $\hat{G}$ and replies to the challenge query (in the CPA game) by sending the challenge ciphertext (of the indistinguishability game). Note that if $i^* \neq \hat{\imath}$, that is, the commitment $\hat{G}$ doesn't coincide with the transcript of the CPA game, then the hybrid returns 0 anyway. The reduction is formally defined in Algorithm 4.11.

Next, consider any two hybrids H^b_ℓ and $\mathsf{H}^b_{\ell+1}$, $\ell \in [0, \tau - 1]$ and $b \in \{0, 1\}$. Also, assume $\mathcal{P}_{\ell+1}$ results from $\mathcal{P}_\ell$ by placing a pebble on the vertex i_0 (the case when a pebble is removed can be argued analogously). The simulation of H^b_ℓ and $\mathsf{H}^b_{\ell+1}$ is the same except for the (used) re-keys outgoing from i_0: in H^b_ℓ they are all real whereas in $\mathsf{H}^b_{\ell+1}$ they are all fake. By the rules of the pebbling game, the children of i_0 all carry pebbles in the configurations $\mathcal{P}_\ell$ and $\mathcal{P}_{\ell+1}$; therefore the simulation doesn't need to know the corresponding secret keys. This means that we can prove indistinguishability of H^b_ℓ and $\mathsf{H}^b_{\ell+1}$ from weak key-privacy: the reduction embeds the (key-privacy) challenge public keys $\mathtt{pk}_0, \ldots, \mathtt{pk}_{\delta_{out}}$ at i_0 and its children, and uses the challenge re-keys $\mathtt{rk}_{0,1}, \ldots, \mathtt{rk}_{0,\delta_{out}}$ to simulate the re-key oracle for queries from i_0 to its children. The reduction is formally defined in Algorithm 4.12. (Note that the simulation of the reduction in Algorithm 4.12 is perfect: if the commitment $\hat{G}$ does not match with the

Algorithm 4.10: Template for generating fully selective PRE-CPA hybrids given a pebbling configuration. All the oracles are defined like in Algorithm 4.3.

Hybrid $\mathsf{H}^b_\ell(1^\lambda, 1^L, N)$

Obtain the challenge graph $\hat{G} \in \mathcal{G}(N, \delta_{out}, D)$ from A

Compute $\mathcal{P}_\ell \leftarrow \mathsf{P}(\hat{G}, \ell)$ `// The ℓth pebbling configuration`

Set $\mathcal{C}, \mathcal{E} = \emptyset$ `// C stores corrupt keys, E re-keys and re-encryptions`

$\mathtt{pp} \leftarrow \mathsf{PRE.S}(1^\lambda, 1^L)$, $(\mathtt{pk}_1, \mathtt{sk}_1), \ldots, (\mathtt{pk}_N, \mathtt{sk}_N) \leftarrow \mathsf{PRE.K}(\mathtt{pp})$

$\forall i \in \mathcal{P}_\ell, \forall j \in \mathsf{children}_{\hat{G}}(i)$: $\mathtt{rk}_{i,j} \leftarrow \mathsf{RK}^*(\mathtt{pp}, \mathtt{pk}_j)$ `// Fake re-keys`

$\forall i \in \mathcal{P}_\ell, \forall j \in [1, N] \setminus \{\mathsf{children}_{\hat{G}}(i) \cup i\}$: $\mathtt{rk}_{i,j} \leftarrow \mathsf{PRE.RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$ `// Real re-keys`

$\forall i \in [1, N] \setminus \mathcal{P}_\ell, \forall j \neq i$: $\mathtt{rk}_{i,j} \leftarrow \mathsf{PRE.RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$ `// Real re-keys`

$b' \leftarrow \mathsf{A}^{(\mathtt{corrupt},\cdot),(\mathtt{rekey},\cdot,\cdot),(\mathtt{reencrypt},\cdot,\cdot,\cdot),(\mathtt{challenge},\cdot,\cdot,\cdot)}(\mathtt{pp}, \mathtt{pk}_1, \ldots, \mathtt{pk}_N)$

if A *made call* $(\mathtt{challenge}, i^*, \cdot, \cdot)$ *for some* i^* `// Check abort conditions`

then

  if $\exists\, i \in \mathcal{C}$: i^* *is connected to* i *in* $([1, N], \mathcal{E})$ **then**

    return 0

if $\hat{G}$ *is the subgraph of* $([1, N], \mathcal{E})$ *reachable from* i^* **then**

  return b'

return 0

transcript, it returns 0; else, we have $\hat{\imath} = i^*$ and by definition of the pebbling, i_0 is reachable from $\hat{\imath} = i^*$ and so are its children $i_1, \ldots, i_{\delta_{out}}$. If the adversary corrupts any of these, then the game returns 0.)

In summary, we get a sequence of hybrids $\mathrm{SEL}_\mathcal{G}[\mathsf{CPA}^0, \mathsf{w}] = \mathsf{H}^0_0, \ldots, \mathsf{H}^0_\tau, \mathsf{H}^1_\tau, \ldots, \mathsf{H}^1_0 = \mathrm{SEL}_\mathcal{G}[\mathsf{CPA}^1, \mathsf{w}]$, where each pair of subsequent hybrids can be proven indistinguishable. Security in the fully selectivised CPA game follows by Lemma 22. We state this formally in Lemma 10 below.

Lemma 10 (Security against fully selectivised PRE-CPA)**.** *Consider the sequence of hybrids* $\mathsf{H}^0_0, \ldots, \mathsf{H}^0_\tau, \mathsf{H}^1_\tau, \ldots, \mathsf{H}^1_0$*, where* H^b_ℓ *is defined in Algorithm 4.10 using the pebbling configuration* $\mathcal{P}_\ell \leftarrow \mathsf{P}(\hat{G}, \ell)$*, where* P *is a reversible source-pebbling strategy (cf. Definition 25).* H^b_0 *is the fully selectivised game of* CPA^b*: i.e.,* $\mathsf{H}^b_0 = \mathrm{SEL}_\mathcal{G}[\mathsf{CPA}^b, \mathsf{w}]$ *where* w *extracts the challenge graph (subgraph reachable from the challenge vertex) from the transcript. Moreover, if the adversary makes at most* Q_RE *re-encryption queries, then a PRE scheme that is* (t_1, ϵ_1)*-indistinguishable and* $(t_2, \epsilon_2, \delta_{out})$*-weakly key-private is* (t, ϵ)*-secure against fully selectivised* PRE-CPA *restricted to challenge graphs in* $\mathcal{G}(N, \delta_{out}, D)$ *with*

$$t := \min(t_1, t_2) - t_\mathsf{CPA} \quad \textit{and} \quad \epsilon := \epsilon_1 + 2\tau \cdot \epsilon_2,$$

where $t_\mathsf{CPA} \approx \mathsf{O}(t_\mathsf{P} + N^2 \cdot t_\mathsf{RK} + Q_\mathsf{RE} \cdot t_\mathsf{RE})$ *denotes the complexity of simulating the CPA game.*

PRE-CPA-security follows from random guessing (Section 3.2, Theorem 1) but with a security loss of 2^{N^2}, where N^2 is an upper bound on the number of bits required to encode the challenge subgraph:

Corollary 3 (PRE-CPA-security by random guessing)**.** *A PRE scheme that is* (t_1, ϵ_1)*-indistinguishable and* $(t_2, \epsilon_2, \delta_{out})$*-weakly key-private is* (t, ϵ)*-secure against* PRE-CPA *restricted*

Algorithm 4.11: The reduction showing that the hybrids H^0_τ and H^1_τ are indistinguishable by indistinguishability of ciphertexts.

Reduction $\mathsf{R}^{(\mathsf{IND}.\mathtt{challenge},\cdot,\cdot)}_\tau(\mathrm{pp}^*, \mathrm{pk}^*)$ // pk^* the challenge public key

1 Obtain the challenge graph $\hat{G} \in \mathcal{G}(N, \delta_{out}, D)$ from A

2 Compute $\mathcal{P}_\tau \leftarrow \mathsf{P}(\hat{G}, \tau)$ // The τth pebbling configuration

3 Set $\mathcal{C}, \mathcal{E} = \emptyset$ // $\mathcal{C}$ stores corrupt keys, $\mathcal{E}$ re-keys and re-encryptions

4 Let $\hat{\imath}$ be the source of $\hat{G}$, set $\mathrm{pk}_{\hat{\imath}} := \mathrm{pk}^*$ // Embed challenge public key

5 $(\mathrm{pk}_1, \mathrm{sk}_1), \ldots, (\mathrm{pk}_{\hat{\imath}-1}, \mathrm{sk}_{\hat{\imath}-1}), (\mathrm{pk}_{\hat{\imath}+1}, \mathrm{sk}_{\hat{\imath}+1}), \ldots, (\mathrm{pk}_N, \mathrm{sk}_N) \leftarrow \mathsf{PRE.K}(\mathrm{pp}^*)$

6 $\forall i \in \mathcal{P}_\tau, \forall j \in \mathsf{children}_{\hat{G}}(i)$: $\mathrm{rk}_{i,j} \leftarrow \mathsf{RK}^*(\mathrm{pp}, \mathrm{pk}_j)$ // Fake re-keys

7 $\forall i \in \mathcal{P}_\tau, \forall j \in [1, N] \setminus \{\mathsf{children}_{\hat{G}}(i) \cup i\}$: $\mathrm{rk}_{i,j} \leftarrow \mathsf{PRE.RK}((\mathrm{pk}_i, \mathrm{sk}_i), \mathrm{pk}_j)$ // Real re-keys

8 $\forall i \in [1, N] \setminus \mathcal{P}_\tau, \forall j \neq i$: $\mathrm{rk}_{i,j} \leftarrow \mathsf{PRE.RK}((\mathrm{pk}_i, \mathrm{sk}_i), \mathrm{pk}_j)$ // Real re-keys

9 $b' \leftarrow \mathsf{A}^{(\mathtt{corrupt},\cdot),(\mathtt{rekey},\cdot,\cdot),(\mathtt{reencrypt},\cdot,\cdot,\cdot),(\mathtt{challenge},\cdot,\cdot,\cdot)}(\mathrm{pp}^*, \mathrm{pk}_1, \ldots, \mathrm{pk}_N)$

10 **if** A *made call* $(\mathtt{challenge}, i^*, \cdot, \cdot)$ *for some* i^* // Check abort conditions

11 **then**

12 | **if** $\exists\, i \in \mathcal{C} : i^*$ *is connected to* i *in* $([1, N], \mathcal{E})$ **then**

13 | | **return** 0

14 **if** $\hat{G}$ *is the subgraph of* $([1, N], \mathcal{E})$ *reachable from* i^* **then**

15 | **return** b'

16 **return** 0

Oracles `rekey` and `reencrypt` are defined like in Algorithm 4.3.

Oracle $(\mathtt{corrupt}, i)$

17 **if** $i = \hat{\imath}$ **then**

18 | HALT: R_τ returns 0 // Commitment $\hat{G}$ doesn't match or i^* corrupted

19 Add i to $\mathcal{C}$ and **return** sk_i

Oracle $(\mathtt{challenge}, i^*, (\mathrm{m}^*_0, \mathrm{m}^*_1), l^*)$ // Single access

20 $(\mathrm{c}_{i^*}, l^*) \leftarrow \mathsf{IND}.\mathtt{challenge}((\mathrm{m}^*_0, \mathrm{m}^*_1), l^*)$ // Embed challenge ciphertext

21 **return** (c_{i^*}, l^*)

to challenge graphs in $\mathcal{G}(N, \delta_{out}, D)$*, where*

$$t := \min(t_1, t_2) - t_{\mathsf{CPA}} - t_{\mathcal{G}} \quad \textit{and} \quad \epsilon := (\epsilon_1 + 2\tau \cdot \epsilon_2) \cdot 2^{N^2}.$$

Partially Selective Hybrids.

In hybrid H^b_ℓ described in Algorithm 4.10, we observe that not all information on the committed recoding graph $\hat{G}$ is actually required for the simulation. In fact, only the pebbling configuration $\mathcal{P}_\ell$ is required to simulate the hybrid: re-keys are only required once a corresponding re-key or a re-encrypt query is issued; for a pebbled node, such queries lead to an edge added in $\mathcal{E}$; thus the re-key is simulated (while the "not-queried" re-keys are never used during the experiment).

In addition to the pebbling configuration $\mathcal{P}_\tau$, the reduction from ciphertext indistinguishability (cf. Algorithm 4.11) also needs to know the challenge vertex $\hat{\imath}$ in order to embed the challenge

Algorithm 4.12: The reduction showing that the hybrids H^b_ℓ and $\mathsf{H}^b_{\ell+1}$, for $\ell \in [0, \tau-1]$ and $b \in \{0,1\}$, are indistinguishable by weak key-privacy.

Reduction $\mathsf{R}^b_\ell(\mathtt{pp}^*, \mathtt{pk}^*_0, \ldots, \mathtt{pk}^*_{\delta_{out}}, \mathtt{rk}^*_{0,1}, \ldots, \mathtt{rk}^*_{0,\delta_{out}})$;

Obtain the challenge graph $\hat{G} \in \mathcal{G}(N, \delta_{out}, D)$ from A;

Compute $\mathcal{P}_\ell \leftarrow \mathsf{P}(\hat{G}, \ell)$, $\mathcal{P}_{\ell+1} \leftarrow \mathsf{P}(\hat{G}, \ell+1)$; // The ℓth and $(\ell+1)$th pebbling configurations

Set $\mathcal{C}, \mathcal{E} = \emptyset$; // $\mathcal{C}$ stores corrupt keys, $\mathcal{E}$ re-keys and re-encryptions

$i_0 := \mathcal{P}_\ell \Delta \mathcal{P}_{\ell+1}$, $i_1, \ldots, i_{\delta_{out}} := \mathsf{children}_{\hat{G}}(i_0)$; // i_0 the pebbled/unpebbled vertex

$\forall k \in [0, \delta_{out}]$: $\mathtt{pk}_{i_k} := \mathtt{pk}^*_k$; // Embed the challenge public keys

$\forall k \in [1, N] \setminus \{i_0, \ldots, i_{\delta_{out}}\}$: $(\mathtt{pk}_k, \mathtt{sk}_k) \leftarrow \mathsf{PRE.K}(\mathtt{pp}^*)$; // Fresh keys

$\forall k \in [1, \delta_{out}] : \mathtt{rk}_{i_0, i_k} := \mathtt{rk}^*_{0,k}$; // Embed challenge re-keys

$\forall i \in \mathcal{P}_\ell \setminus \{i_0\}, \forall j \in \mathsf{children}_{\hat{G}}(i)$: $\mathtt{rk}_{i,j} \leftarrow \mathsf{RK}^*(\mathtt{pp}^*, \mathtt{pk}_j)$; // Fake re-keys

$\forall i \in \mathcal{P}_\ell, \forall j \in [1, N] \setminus \{\mathsf{children}_{\hat{G}}(i) \cup i\}$: $\mathtt{rk}_{i,j} \leftarrow \mathsf{PRE.RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$; // Real re-keys

$\forall i \in [1, N] \setminus (\mathcal{P}_\ell \cup \{i_0\}), \forall j \neq i$: $\mathtt{rk}_{i,j} \leftarrow \mathsf{PRE.RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$; // Real re-keys

$b' \leftarrow \mathsf{A}^{(\mathtt{corrupt},\cdot),(\mathtt{rekey},\cdot,\cdot),(\mathtt{reencrypt},\cdot,\cdot,\cdot),(\mathtt{challenge},\cdot,\cdot,\cdot)}(\mathtt{pp}^*, \mathtt{pk}_1, \ldots, \mathtt{pk}_N)$;

if A *made call* $(\mathtt{challenge}, i^*, \cdot, \cdot)$ *for some* i^* ; // Check abort conditions

then

  if $\exists\, i \in \mathcal{C} : i^*$ *is connected to* i *in* $([1, N], \mathcal{E})$ **then**

    return 0;

if $\hat{G}$ *is the subgraph of* $([1, N], \mathcal{E})$ *reachable from* i^* **then**

  return b';

return 0;

Oracles `rekey`, `reencrypt` and `challenge` are defined like in Algorithm 4.3;

Oracle $(\mathtt{corrupt}, i)$;

if $i \in \{i_0, \ldots, i_{\delta_{out}}\}$ **then**

  HALT: R_τ returns 0 ; // Commitment $\hat{G}$ doesn't match or i^* connected to i

Add i to $\mathcal{C}$ and **return** $\mathtt{sk}_i$;

public key. The reduction from weak key-privacy (cf. Algorithm 4.12) requires, in addition to $\mathcal{P}_\ell$, the vertex that is pebbled or unpebbled in $\mathcal{P}_{\ell+1}$ (i.e., the vertex i_0) and its children, so it can embed its challenge public keys and re-keys.

To sum up, two consecutive hybrids H^b_ℓ and $\mathsf{H}^b_{\ell+1}$ can be shown to be indistinguishable using a lot less information than what the adversary commits to. We thus have the following:

Lemma 11 (Partially selectivised hybrids)**.** *Let* $\mathcal{P}_0, \ldots, \mathcal{P}_\tau$ *and* $\mathsf{H}^0_0, \ldots, \mathsf{H}^0_\tau$, $\mathsf{H}^1_\tau, \ldots, \mathsf{H}^1_0$ *be defined as in Lemma 10, and let* σ *denote the space complexity of the pebbling sequence. Then, for* $\ell \in [0, \tau-1]$ *and* $b, \beta \in \{0,1\}$*,*

$$\mathsf{H}^b_{\ell+\beta} \equiv \mathrm{SEL}_{\mathcal{U}\to\mathcal{G}}[\hat{\mathsf{H}}^b_{\ell,\beta}, \mathsf{w}, \mathsf{h}_\ell] \quad \textit{and} \quad \mathsf{H}^b_\tau \equiv \mathrm{SEL}_{\mathcal{U}\to\mathcal{G}}[\hat{\mathsf{H}}^b_{\tau,0}, \mathsf{w}, \mathsf{h}_\tau],$$

where $\hat{\mathsf{H}}^b_{\ell,\beta}$ *is defined in Algorithm 4.13 (see also Figure 4.2) and* w *extracts the challenge*

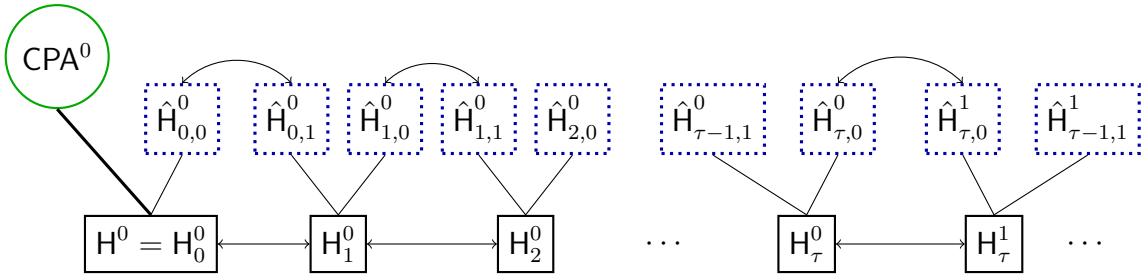


Figure 4.2: Diagram showing the partially selectivised hybrids for PRE-CPA.

graph from the transcript (as in Lemma 10). For $\ell \in [0, \tau - 1]$, h_ℓ is the function that extracts the pebbling configuration $\mathcal{P}_\ell$, the pebbled/unpebbled vertex in $\mathcal{P}_{\ell+1}$ and its children; h_τ extracts the pebbling configuration $\mathcal{P}_\tau$ and the challenge node i^*. Thus, $\mathcal{U}$ corresponds to the set $\mathcal{V}^{\sigma+\delta_{out}+1}$.

The tighter bound for PRE-CPA-security now results by applying Theorem 2:

Theorem 7 (Main, PRE-CPA security)**.** *Let σ and τ denote, respectively, the pebbling space and time complexity for the class $\mathcal{G}(N, \delta_{out}, D)$. Then a PRE scheme that is (t_1, ϵ_1)-indistinguishable and $(t_2, \epsilon_2, \delta_{out})$-weakly key-private is (t, ϵ)-*PRE-CPA*-secure restricted to challenge graphs in $\mathcal{G}(N, \delta_{out}, D)$, where*

$$t := \min(t_1, t_2) - t_{\mathsf{CPA}} - t_{\mathcal{G}} \quad \textit{and} \quad \epsilon := (\epsilon_1 + 2\tau \cdot \epsilon_2) \cdot N^{\sigma+\delta_{out}+1}.$$

4.4.2 Adaptive Security Against Honest-Reencryption Attack

Cohen [Coh19] showed that if a PRE scheme is re-encryption-simulatable then selective security against HRA reduces to showing selective security against CPA. We now consider such a reduction in the adaptive setting. This is not immediate because the reduction in [Coh19] simulates all re-encryption queries from honest keys to corrupt keys. This works in the selective setting, where the set of corrupt users is known in advance, but not in the adaptive setting.

The recoding graphs $([1, N], \mathcal{E})$ for CPA and HRA are defined differently: for HRA, only re-encryptions of the challenge ciphertexts lead to edges in $\mathcal{E}$, whereas for CPA all re-encryption queries do. To prove that CPA implies HRA we need to define a reduction playing the CPA game and simulating the HRA game for an adversary. It cannot forward the adversary's re-encryption queries to its own challenger, as this might create edges in the CPA game, but not in the HRA game. An adversary that then corrupts the target key of such a re-encryption query might still win the HRA game, while the reduction loses the CPA game.

Assuming source-hiding, the reduction could answer all re-encryption queries by fresh encryptions. But then every re-encryption of the challenge ciphertext would also be freshly encrypted, meaning that the reduction needs to make multiple challenge queries. This would require a multi-challenge notion of CPA and thus worsen the security guarantees; we proceed thus differently. Instead of replacing all re-encryptions by fresh encryptions, the reduction only replaces those that do *not* concern the challenge ciphertext, while still forwarding re-encryption queries of (derivatives of) the challenge ciphertext to its own re-encryption oracle. The vertices created in the HRA game correspond then precisely to those created in the CPA game and the adversary's success probability translates directly to that of the reduction.

Algorithm 4.13: Partially selectivised hybrids. For $\ell \in [0, \tau - 1]$ and $b, \beta \in \{0,1\}$: $\mathsf{H}^b_{\ell+\beta} = \mathrm{SEL}_{\mathcal{U}\to\mathcal{G}}[\hat{\mathsf{H}}^b_{\ell,\beta}, \mathsf{w}, \mathsf{h}_\ell]$ and $\mathsf{H}^b_\tau = \mathrm{SEL}_{\mathcal{U}\to\mathcal{G}}[\hat{\mathsf{H}}^b_{\tau,0}, \mathsf{w}, \mathsf{h}_\tau]$. Moreover, $\mathcal{U} := \mathcal{V}^{\sigma+\delta_{out}+1}$. Note that the sampling of re-keys is deferred to the actual calls.

Hybrid $\mathsf{H}^b_{\ell+\beta}$

Obtain the challenge graph $\hat{G} \in \mathcal{G}(N, \delta_{out}, D)$ from A and let $\hat{\imath}$ be its source
Compute $\mathcal{P}_\ell \leftarrow \mathsf{P}(\hat{G}, \ell)$, $\mathcal{P}_{\ell+1} \leftarrow \mathsf{P}(\hat{G}, \ell+1)$ // ℓth, $(\ell+1)$th configuration
$i_0 := \mathcal{P}_\ell \Delta \mathcal{P}_{\ell+1}$, $i_1, \ldots, i_{\delta_{out}} := \mathsf{children}(i_0, \hat{G})$ // i_0 pebbled/unpebbled vertex
if $t < \tau$ **then** $b' \leftarrow \hat{\mathsf{H}}^b_{\ell,\beta}(\mathcal{P}_\ell, \{i_0, \ldots, i_{\delta_{out}}\})$ // Key-privacy hybrid
else $b' \leftarrow \hat{\mathsf{H}}^b_{\tau,0}(\mathcal{P}_\tau, \{\hat{\imath}, \bot, \ldots, \bot\})$ // indistinguishability hybrid
if $\hat{G}$ *is the subgraph of* $([1,N], \mathcal{E})$ *reachable from* i^* **then return** b'
return 0

$\hat{\mathsf{H}}^b_{\ell,\beta}(\mathcal{P}_\ell, \{i_0, \ldots, i_{\delta_{out}}\})$

Set $\mathcal{C}, \mathcal{E} = \emptyset$ // $\mathcal{C}$ stores corrupt keys, $\mathcal{E}$ re-keys and re-encryptions
if $t < \tau$ **then**
  if $i_0 \in \mathcal{P}_\ell$ **then** $\mathcal{P}_{\ell+1} := \mathcal{P}_\ell \setminus \{i_0\}$
  else $\mathcal{P}_{\ell+1} := \mathcal{P}_\ell \cup \{i_0\}$
$\mathtt{pp} \leftarrow \mathsf{PRE.S}(1^\lambda, 1^L)$, $(\mathtt{pk}_1, \mathtt{sk}_1), \ldots, (\mathtt{pk}_N, \mathtt{sk}_N) \leftarrow \mathsf{PRE.K}(\mathtt{pp})$
$\forall i, j \in [1,N], i \neq j : \mathtt{rk}_{i,j} = \bot$ // Delay re-key generation till the query
$b' \leftarrow \mathsf{A}^{(\mathtt{corrupt},\cdot),(\mathtt{rekey},\cdot,\cdot),(\mathtt{reencrypt},\cdot,\cdot,\cdot),(\mathtt{challenge},\cdot,\cdot,\cdot)}(\mathtt{pp}, \mathtt{pk}_1, \ldots, \mathtt{pk}_N)$
if A *made call* $(\mathtt{challenge}, i^*, \cdot, \cdot)$ *for some* i^* **then** // Check abort conditions
  if $\exists\, i \in \mathcal{C} : i^*$ *is connected to* i *in* $([1,N], \mathcal{E})$ **then return** 0
return b'

Oracles `corrupt` and `challenge` are defined like in Algorithm 4.3.

Oracle $(\mathtt{rekey}, i, j)$

if $rk_{i,j} = \bot$ **then** // Re-key not generated
  if $i \in \mathcal{P}_{\ell+\beta}$ **then** $\mathtt{rk}_{i,j} \leftarrow \mathsf{RK}^*(\mathtt{pp}, \mathtt{pk}_j)$ // Fake re-key
  else $\mathtt{rk}_{i,j} \leftarrow \mathsf{RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$ // Real re-key
Add (i,j) to $\mathcal{E}$ // Add to recoding graph
return $\mathtt{rk}_{i,j}$

Oracle $(\mathtt{reencrypt}, i, j, (\mathtt{c}_i, l))$

if $rk_{i,j} = \bot$ **then** // Re-key not generated
  if $i \in \mathcal{P}_{\ell+\beta}$ **then** $\mathtt{rk}_{i,j} \leftarrow \mathsf{RK}^*(\mathtt{pp}, \mathtt{pk}_j)$ // Fake re-key
  else $\mathtt{rk}_{i,j} \leftarrow \mathsf{RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$ // Real re-key
Add (i,j) to $\mathcal{E}$ // Add to recoding graph
return $(\mathtt{c}_j, l+1) \leftarrow \mathsf{PRE.RE}(\mathtt{rk}_{i,j}, \mathtt{pk}_i, \mathtt{pk}_j, (\mathtt{c}_i, l))$

HRA from CPA and Source-Hiding.

We start by defining, in Algorithm 4.14, the intermediate game shHRA^b just discussed. It proceeds like HRA^b, except that all re-encryption queries which do *not* re-encrypt the challenge

Algorithm 4.14: Intermediate game shHRA in the proof of HRA.

Hybrid shHRA^b

Set $\mathcal{C}, \mathcal{L}, \mathcal{L}^*, \mathcal{E} = \emptyset$ and $ctr = 0$ // As in Algorithm 4.2, $\mathcal{E}$...edges of recoding graph

$\mathtt{pp} \leftarrow \mathsf{PRE.S}(1^\lambda, 1^L)$, $(\mathtt{pk}_1, \mathtt{sk}_1), \ldots, (\mathtt{pk}_N, \mathtt{sk}_N) \leftarrow \mathsf{PRE.K}(\mathtt{pp})$

$\forall i, j \in [1, N], i \neq j : \mathtt{rk}_{i,j} \leftarrow \mathsf{PRE.RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$

$b' \leftarrow \mathsf{A}^{(\mathtt{corrupt},\cdot),(\mathtt{rekey},\cdot,\cdot),(\mathtt{encrypt},\cdot,\cdot),(\mathtt{reencrypt},\cdot,\cdot,\cdot),(\mathtt{challenge},\cdot,\cdot,\cdot)}(\mathtt{pp}, \mathtt{pk}_1, \ldots, \mathtt{pk}_N)$

if A *made call* $(\mathtt{challenge}, i^*, \cdot, \cdot)$ *for some* i^* **then** // Check abort conditions

  if $\exists\, i \in \mathcal{C} : i^*$ *is connected to* i **then**

    return 0

return b'

Oracles `corrupt`, `rekey`, `encrypt`, `challenge` are defined as in Algorithm 4.4.

Oracle $(\mathtt{reencrypt}, i, j, k)$

Retrieve $(k, i, \mathtt{m}, (\mathtt{c}_i, l))$ from $\mathcal{L}$ and increment *ctr*

if $k \notin \mathcal{L}^*$ **then** // Not a re-encryption of challenge ciphertext

  $(\mathtt{c}_j, l+1) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_j, (\mathtt{m}, l+1))$ // Simulate re-encryption

else

  $(\mathtt{c}_j, l+1) \leftarrow \mathsf{PRE.RE}(\mathtt{rk}_{i,j}, \mathtt{pk}_i, \mathtt{pk}_j, (\mathtt{c}_i, l))$ // Real re-encryption

  Add *ctr* to $\mathcal{L}^*$ and add (i, j) to $\mathcal{E}$ // $\mathtt{c}_j$ derived from challenge

Add $(ctr, j, \mathtt{m}, (\mathtt{c}_j, l+1))$ to $\mathcal{L}$

return $(\mathtt{c}_j, l+1)$

ciphertext are simulated. The games HRA^b and shHRA^b are shown to be indistinguishable assuming source-hiding by a standard hybrid argument (without pebbling) which includes a moderate amount of guessing: when replacing a re-encryption by a fresh encryption, it guesses the two concerned users and which ciphertext will be re-encrypted. This loses a factor of $N(N-1)(Q_{\mathsf{RE}} + Q_{\mathsf{E}})$ in the distinguishing advantage. As there are Q_{RE} hybrids, we get the following:

Lemma 12. *If a PRE scheme is* (t_3, ϵ_3)*-source-hiding as per Definition 24 then* HRA^b *and* shHRA^b *are* (t, ϵ)*-indistinguishable, where*

$$t := t_3 - t_{\mathsf{HRA}} \quad and \quad \epsilon := N(N-1)(Q_{\mathsf{E}} + Q_{\mathsf{RE}})Q_{\mathsf{RE}} \cdot \epsilon_3$$

where Q_{E} *and* Q_{RE} *are upper bounds on the number of the adversary's encryption and re-encryption queries.*

Proof. We define a sequence of intermediate hybrids $\mathsf{shHRA}^b_0, \ldots, \mathsf{shHRA}^b_{Q_{\mathsf{RE}}}$ between HRA^b and shHRA^b where certain re-encrypt queries are simulated by computing fresh encryptions. In particular, the simulation in shHRA^b_q is similar to that in HRA^b, except that the first q queries $(\mathtt{reencrypt}, \cdot, \cdot, k)$ with $k \notin \mathcal{L}^*$ are replied by fresh encryptions. Therefore, we have

$$\mathsf{shHRA}^b_0 \equiv \mathsf{HRA}^b \quad \text{and} \quad \mathsf{shHRA}^b_{Q_{\mathsf{RE}}} \equiv \mathsf{shHRA}^b.$$

We show that two neighbouring hybrids shHRA^b_{q-1} and shHRA^b_q are indistinguishable assuming that the PRE scheme is source-hiding. The reduction receives a (source-hiding) challenge

$(\mathtt{pp}, (\mathtt{pk}, \mathtt{sk}), (\mathtt{pk}', \mathtt{sk}'), \mathtt{rk})$ and has (one-time) access to an oracle $(\mathsf{SH}.\mathtt{challenge}, \cdot, \cdot)$. The reduction proceeds as follows:

1. It first makes a guess $(i^*, j^*, k^*) \in [1, N] \times ([1, N] \setminus \{i^*\}) \times [1, Q_{\mathsf{E}} + q - 1]$ that the q-th re-encryption query will be of the form $(\mathtt{reencrypt}, i^*, j^*, k^*)$
2. It simulates game shHRA^b_{q-1} setting $(\mathtt{pk}_{i^*}, \mathtt{sk}_{i^*}) := (\mathtt{pk}, \mathtt{sk})$, $(\mathtt{pk}_{j^*}, \mathtt{sk}_{j^*}) := (\mathtt{pk}', \mathtt{sk}')$ and $\mathtt{rk}_{i^*,j^*} := \mathtt{rk}$.
3. When the adversary makes the k^*-th encrypt or re-encrypt query, the reduction does the following:
   a) $(\mathtt{encrypt}, i, (\mathtt{m}, l))$: if $i \neq i^*$ abort; else query $(\mathsf{SH}.\mathtt{challenge}, \mathtt{m}, l)$ to receive $(\mathtt{c}, \mathtt{c}')$; reply $(\mathtt{c}, l)$ after adding the new entry to $\mathcal{L}$.
   b) $(\mathtt{reencrypt}, i, j, k)$: if $k \in \mathcal{L}^*$ then proceed as in Algorithm 4.4. Otherwise: if $j \neq i^*$ abort; else retrieve $(k, i, \mathtt{m}, (\mathtt{c}_i, l-1))$ from $\mathcal{L}$, query $(\mathsf{SH}.\mathtt{challenge}, \mathtt{m}, l)$ to receive $(\mathtt{c}, \mathtt{c}')$; reply $(\mathtt{c}, l)$ after adding the new entry to $\mathcal{L}$.
4. When the adversary makes the q-th re-encrypt query $(\mathtt{reencrypt}, i, j, k)$, the reduction aborts if $(i, j, k) \neq (i^*, j^*, k^*)$. Otherwise it replies $(\mathtt{c}', l+1)$, with $\mathtt{c}'$ received on its $\mathsf{SH}.\mathtt{challenge}$ query.
5. If the reduction aborted the simulation, it returns a random bit, otherwise it returns the adversary's output bit b'.

Assuming the reduction's guess is right, if the $\mathtt{c}'$ returned by the source-hiding challenger is a re-encryption then the reduction simulated shHRA^b_{q-1} while if $\mathtt{c}'$ was a fresh encryption, it simulated shHRA^b_q. □

We next show that shHRA^0 and shHRA^1 are indistinguishable assuming CPA^0 and CPA^1 are. To do so, we construct a reduction R in Algorithm 4.15, which simulates game shHRA to an adversary A (denoted R[A]); the reduction itself attacks game CPA. It is easy to see that R perfectly simulates the oracles of shHRA to A and that at the end of the game the sets $\mathcal{C}$ and the graphs $([1, N], \mathcal{E})$ which were implicitly defined by the adversary's oracle calls in the simulated HRA game and the reduction's calls in the CPA game are the same. The CPA game thus returns the reduction's output b' (which is A's output) whenever the shHRA game would. Thus, we have:

$$\langle \mathsf{shHRA}^b, \mathsf{R}[\mathsf{A}] \rangle \equiv \langle \mathsf{CPA}^b, \mathsf{A} \rangle, \tag{4.1}$$

that is, the two games are equally distributed. (This can also be seen by replacing A in the definition of CPA (Algorithm 4.3) by the code of $\mathsf{R}[\mathsf{A}']$ (Algorithm 4.15), which yields game HRA (Algorithm 4.4) played by A'.)

Combining Lemma 12 and (4.1), we get that HRA^b and CPA^b are $(t_3 - t_{\mathsf{HRA}}, N(N-1)(Q_{\mathsf{E}} + Q_{\mathsf{RE}})Q_{\mathsf{RE}} \cdot \epsilon_3)$-indistinguishable, and together with Theorem 7 this finally yields:

Theorem 8 (main, PRE-HRA security)**.** *Let σ and τ denote, respectively, an upper bound on space and time complexity for the class $\mathcal{G} = \mathcal{G}(N, \delta_{out}, D)$. Then a PRE scheme that is (t_1, ϵ_1)-indistinguishable, $(t_2, \epsilon_2, \delta_{out})$-weakly key-private and (t_3, ϵ_3)-source-hiding is* (t, ϵ)-PRE-HRA*-secure restricted to challenge graphs in $\mathcal{G}$, where $t := \min(t_1, t_2, t_3) - t_{\mathsf{HRA}} - t_{\mathcal{G}}$ and*

$$\epsilon := 2N(N-1)(Q_{\mathsf{E}} + Q_{\mathsf{RE}})Q_{\mathsf{RE}} \cdot \epsilon_3 + N^{\sigma+\delta_{out}+1}\Big(\epsilon_1 + 2\tau \cdot \epsilon_2\Big).$$

Algorithm 4.15: The reduction relating shHRA^b to CPA^b.

$\mathsf{R}^{b,(\mathsf{CPA}.\mathtt{corrupt},\cdot),(\mathsf{CPA}.\mathtt{rekey},\cdot,\cdot),(\mathsf{CPA}.\mathtt{reencrypt},\cdot,\cdot,\cdot),(\mathsf{CPA}.\mathtt{challenge},\cdot,\cdot,\cdot)}(\mathtt{pp},\mathtt{pk}_1,\ldots,\mathtt{pk}_N)$

Set $\mathcal{L},\mathcal{L}^* = \emptyset$ and $ctr = 0$ `// Stores honestly created ciphertexts`

$b' \leftarrow \mathsf{A}^{(\mathtt{corrupt},\cdot),(\mathtt{rekey},\cdot,\cdot),(\mathtt{encrypt},\cdot,\cdot),(\mathtt{reencrypt},\cdot,\cdot,\cdot),(\mathtt{challenge},\cdot,\cdot,\cdot)}(\mathtt{pp},\mathtt{pk}_1,\ldots,\mathtt{pk}_N)$

return b'

Oracle $(\mathtt{corrupt},i)$

return $\mathtt{sk}_i \leftarrow (\mathsf{CPA}.\mathtt{corrupt},i)$

Oracle $(\mathtt{rekey},i,j)$

return $\mathtt{rk}_{i,j} \leftarrow (\mathsf{CPA}.\mathtt{rekey},i,j)$

Oracle $(\mathtt{encrypt},i,(\mathtt{m},l))$

$c \leftarrow \mathsf{PRE.E}(\mathtt{pk}_i,(\mathtt{m},l))$, increment *ctr* and add $(ctr,i,\mathtt{m},(\mathtt{c},l))$ to $\mathcal{L}$

return c

Oracle $(\mathtt{reencrypt},i,j,k)$

Retrieve $(k,i,\mathtt{m},(\mathtt{c}_i,l))$ from $\mathcal{L}$ and increment *ctr*

if $k \notin \mathcal{L}^*$ **then**

  $(\mathtt{c}_j,l+1) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_j,(\mathtt{m},l+1))$

else

  $(\mathtt{c}_j,l+1) \leftarrow (\mathsf{CPA}.\mathtt{reencrypt},i,j,k)$ and add *ctr* to $\mathcal{L}^*$

Add $(ctr,j,\mathtt{m},(\mathtt{c}_j,l+1))$ to $\mathcal{L}$

return $(\mathtt{c}_j,l+1)$

Oracle $(\mathtt{challenge},i^*,(\mathtt{m}_0^*,\mathtt{m}_1^*),l^*)$ `// Single access`

$(\mathtt{c}_{i^*},l^*) \leftarrow (\mathsf{CPA}.\mathtt{challenge},i^*,(\mathtt{m}_0^*,\mathtt{m}_1^*),l^*)$

Increment *ctr*, add $(ctr,i^*,\mathtt{m}_b^*,(\mathtt{c}_{i^*},l^*))$ to $\mathcal{L}$ and *ctr* to $\mathcal{L}^*$

return $(\mathtt{c}_{i^*},l^*)$

Remark 1. All the schemes that we inspect in Section 4.5 turn out to be *statistically* source-hiding, and therefore ϵ_3 is exponentially small. In such cases, assuming that the adversary is allowed to make only polynomially many queries to the encryption and re-encryption oracle, the term $2N(N-1)(Q_{\mathsf{E}}+Q_{\mathsf{RE}})Q_{\mathsf{RE}}\cdot\epsilon_3$ is negligible and therefore $\epsilon = \mathsf{O}(N^{\sigma+\delta_{out}+1}(\epsilon_1+2\tau\cdot\epsilon_2))$ (just like in CPA).

4.4.3 Corollaries

We calculate concrete bounds to Theorems 7 and 8 for the following families of recoding graphs: arbitrary DAGs in $\mathcal{G}(N,\delta_{out},D)$, complete binary trees $\mathcal{B}(N)$ and chains $\mathcal{C}(N)$. Table 4.2 lists the space and time complexity for these classes (from Lemmas 7, 8 and 9) and the approximate security loss (assuming $\epsilon_1=\epsilon_2=\epsilon'$) that we obtain when substituting these bounds for CPA in Theorem 7. The same bounds hold for HRA if one assumes that Q_{RE} and Q_{E} (i.e., number of queries) are polynomial and $\epsilon_3 = 2^{-\lambda}$ (i.e., the PRE scheme is *statistically* source-hiding).

Family	Bounds		
	Space (σ)	Time (τ)	Security loss ($\approx \epsilon/\epsilon'$)
Arbitrary DAGs $\mathcal{G}(N, \delta_{out}, D)$ (Lemma 7)	$(\delta_{out}+1)\cdot D$	$(2\delta_{out})^D$	$N^{\mathsf{O}(D\cdot\delta_{out})}$
Complete binary trees $\mathcal{B}(N)$ (Lemma 8)	$\log N$	N^2	$N^{\mathsf{O}(\log N)}$
Chains $\mathcal{C}(N)$ (Lemma 9)	$\log N + 1$	$3^{\log N}$	$N^{\mathsf{O}(\log N)}$

Table 4.2: Space and time complexity for different classes of DAGs and approximate security loss implied by Theorem 7.

4.5 Adaptively Secure PRE Schemes

We show that several existing PRE schemes satisfy the requirements in Section 4.3.1 and, therefore, can be proven adaptively secure using Theorems 7 and 8.

4.5.1 Single-Hop Schemes from Bilinear Maps

We start with the unidirectional, *single-hop* schemes based on bilinear maps from [AFGH05] and [ABH09]. The definition of bilinear maps is given below in Definition 27; the hardness assumptions on which security of the constructions is based are then listed in Definitions 29 through 31.

Definition 27 (Bilinear maps)**.** Let BS′ be an algorithm that on input a security parameter 1^λ outputs the description of cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, all of prime order $p \in \Theta(2^\lambda)$, generators $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, and an *asymmetric* cryptographic bilinear map $e' \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ – i.e., e' is efficiently computable, bilinear (i.e., $e'(g_1^a, g_2^b) = e'(g_1, g_2)^{ab}$) and non-degenerate (i.e., for a generator g_1 for $\mathbb{G}_1$ and a generator g_2 for $\mathbb{G}_2$: $e'(g_1, g_2) \neq 1_{\mathbb{G}_T}$).

The algorithm BS for *symmetric* bilinear groups is defined similar to BS′, except for having $\mathbb{G}_1 = \mathbb{G}_2$ and $g_1 = g_2$.

Definition 28 (eDBDH in asymmetric groups [AFGH05])**.** Let $Grp = (p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e') \leftarrow \mathsf{BS}'(1^\lambda)$ and $a, b, c, r \leftarrow \mathbb{Z}_p$. The extended decisional bilinear Diffie-Hellman problem is (t, ϵ)-hard in the asymmetric setting if

$$(g_2^a, g_1^b, g_1^c, e'(g_1, g_2)^{bc^2}, e'(g_1, g_2)^{abc}, Grp) \approx_{(t,\epsilon)} (g_2^a, g_1^b, g_1^c, e'(g_1, g_2)^{bc^2}, e'(g_1, g_2)^r, Grp).$$

The symmetric variant of the above is obtained by replacing $\mathbb{G}_1$ and $\mathbb{G}_2$ by $\mathbb{G}$, and g_1 and g_2 by g:

Definition 29 (eDBDH assumption [AFGH05])**.** Let $Grp := (p, g, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathsf{BS}(1^\lambda)$ and $a, b, c, r \leftarrow \mathbb{Z}_p$. The *extended* decisional bilinear Diffie-Hellman problem is (t, ϵ)-hard if

$$(g^a, g^b, g^c, e(g, g)^{bc^2}, e(g, g)^{abc}, Grp) \approx_{(t,\epsilon)} (g^a, g^b, g^c, e(g, g)^{bc^2}, e(g, g)^r, Grp).$$

Definition 30 (XDH [Sco02, BBS04])**.** Let $Grp = (p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e') \leftarrow \mathsf{BS}'(1^\lambda)$ and $a, b, r \leftarrow \mathbb{Z}_p$. The external Diffie-Hellman problem is (t, ϵ)-hard if the decisional Diffie-Hellman problem is (t, ϵ)-hard in the group $\mathbb{G}_1$, that is

$$(g_1^a, g_1^b, g_1^{ab}, Grp) \approx_{(t,\epsilon)} (g_1^a, g_1^b, g_1^r, Grp).$$

Definition 31 (DLin [BBS04])**.** Let $Grp = (p, g, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathsf{BS}(1^\lambda)$, $a, b, r \leftarrow \mathbb{Z}_p$ and h, f be two random generators of $\mathbb{G}$. The decision linear problem is (t, ϵ)-hard if

$$(h, f, g^a, h^b, f^{a+b}, Grp) \approx_{(t,\epsilon)} (h, f, g^a, h^b, f^r, Grp).$$

1. $\mathsf{S}(1^\lambda)$: $(p, g, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathsf{BS}(1^\lambda)$. Compute $Z = e(g, g)$ and return the public parameters $\mathtt{pp} = ((p, g, \mathbb{G}, \mathbb{G}_T, e), Z)$.
2. $\mathsf{K}(\mathtt{pp})$: Pick $a, b \leftarrow \mathbb{Z}_p$ and set $\mathtt{pk} := (Z^a, g^b)$ and $\mathtt{sk} := (a, b)$. Return the keys $(\mathtt{pk}, \mathtt{sk})$.
3. $\mathsf{RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$: Parse $\mathtt{sk}_i$ as $(a_i, b_i) \in \mathbb{Z}_p^2$ and $\mathtt{pk}_j$ as $(\mathtt{pk}_{j,1}, \mathtt{pk}_{j,2})$. Return the re-key $\mathtt{rk}_{i,j} := \mathtt{pk}_{j,2}^{a_i} = g^{b_j \cdot a_i}$.
4. $\mathsf{E}(\mathtt{pk}, (\mathtt{m}, l))$: Parse $\mathtt{pk}$ as $(\mathtt{pk}_1, \mathtt{pk}_2) \in \mathbb{G}_T \times \mathbb{G}$ and pick $k \leftarrow \mathbb{Z}_p$. If $l = 1$ return the (level-1) ciphertext $((g^k,\ m \cdot \mathtt{pk}_1^k = \mathtt{m} \cdot Z^{ak}),\ 1)$;
otherwise return the (level-2) ciphertext $((e(\mathtt{pk}_2, g)^k,\ \mathtt{m} \cdot Z^k) = (Z^{bk},\ \mathtt{m} \cdot Z^k),\ 2)$.
5. $\mathsf{RE}((\mathtt{c}_i, 1), \mathtt{rk}_{i,j})$: Parse $\mathtt{c}_i$ as $(\mathtt{c}_{i,1}, \mathtt{c}_{i,2}) \in \mathbb{G} \times \mathbb{G}_T$ and return the level-2 ciphertext $((e(\mathtt{rk}_{i,j}, \mathtt{c}_{i,1}),\ \mathtt{c}_{i,2}) = (Z^{b_j \cdot (a_i k)},\ \mathtt{m} \cdot Z^{a_i k}),\ 2)$.
6. $\mathsf{D}((\mathtt{c}, l), \mathtt{sk})$: Parse the secret key $\mathtt{sk}$ as $(a, b) \in \mathbb{Z}_p^2$. Parse a level-1 ciphertext as $(\mathtt{c}_1, \mathtt{c}_2) \in \mathbb{G} \times \mathbb{G}_T$ and return $\mathtt{c}_2 / e(\mathtt{c}_1, g)^a$. Parse a level-2 ciphertext as $(\mathtt{c}_1, \mathtt{c}_2) \in \mathbb{G}_T^2$ and return $\mathtt{c}_2 / c_1^{1/b}$.

Construction 1: Unidirectional, single-hop PRE from [AFGH05]; basis for Construction 2.

The AFGH Scheme.

The original scheme (Construction 1) encrypts messages from $\mathbb{G}_\ell$ and was shown selectively secure against CPA assuming eDBDH, but [AFGH05] did not consider key-privacy. We prove that when instantiated over asymmetric bilinear groups, their scheme is key-private assuming XDH. To additionally make the scheme source-hiding we *rerandomize* the re-encryption algorithm. The modified scheme is given in Construction 2.

Security. If the eDBDH problem is (t_1, ϵ_1)-hard then the original scheme (Construction 1) is $(t_1, 2\epsilon_1)$-sPRE-CPA-secure – and hence has indistinguishable ciphertexts [AFGH05, Theorem 3.1]. They also claim security when instantiated in the asymmetric setting, presumably assuming that the eDBDH problem is hard in the asymmetric setting: this reduction works for Construction 2 too. It moreover satisfies the remaining two properties:

Lemma 13. *Construction 2 is statistically source-hiding.*

Proof. A level-2 ciphertext under $\mathtt{pk}_j$ that results from the re-encryption of a level-1 ciphertext $\mathtt{c}_i = (\mathtt{c}_{i,1}, \mathtt{c}_{i,2})$ under $\mathtt{pk}_i = (\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2})$ is of the form $\mathtt{c}_j = (\mathtt{c}_{j,1}, \mathtt{c}_{j,2})$

$$\begin{aligned} \mathtt{c}_{j,1} &= e'(\mathtt{rk}_{i,j}, \mathtt{c}_{i,1} \cdot g_2^{k'}) = e'(g_1^{b_j \cdot a_i}, g_2^k \cdot g_2^{k'}) = e'(g_1^{b_j}, g_2)^{a_i \cdot (k + k')} \\ \mathtt{c}_{j,2} &= c_{i,2} \cdot \mathtt{pk}_{i,1}^{k'} = \mathtt{m} \cdot Z^{a_i \cdot k} \cdot Z^{a_i \cdot k'} = \mathtt{m} \cdot Z^{a_i \cdot (k + k')}. \end{aligned} \tag{4.2}$$

From (4.2) it is clear that the distribution of the re-encrypted ciphertext is statistically close to a fresh level-2 ciphertext under $\mathtt{pk}_j$. □

1. $\mathsf{S}(1^\lambda)$: $\boxed{(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e') \leftarrow \mathsf{BS}'(1^\lambda)}$. Compute $Z = e'(g_1, g_2)$ and return the public parameters $\mathtt{pp} = ((p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e'), Z)$.
2. $\mathsf{K}(\mathtt{pp})$: Pick $a, b \leftarrow \mathbb{Z}_p$ and set $\mathtt{pk} := (Z^a, g_1^b)$ and $\mathtt{sk} := (a, b)$. Return the keys $(\mathtt{pk}, \mathtt{sk})$.
3. $\mathsf{RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$: Parse $\mathtt{sk}_i$ as $(a_i, b_i) \in \mathbb{Z}_p^2$ and $\mathtt{pk}_j$ as $(\mathtt{pk}_{j,1}, \mathtt{pk}_{j,2}) \in \mathbb{G}_T \times \mathbb{G}_1$. Return the re-key $\mathtt{rk}_{i,j} := \mathtt{pk}_{j,2}^{a_i} = g_1^{b_j \cdot a_i}$.
4. $\mathsf{E}(\mathtt{pk}, (\mathtt{m}, l))$: Parse $\mathtt{pk}$ as $(\mathtt{pk}_1, \mathtt{pk}_2) \in \mathbb{G}_T \times \mathbb{G}_1$ and pick $k \leftarrow \mathbb{Z}_p$. If $l = 1$ return the (level-1) ciphertext $((g_2^k, m \cdot \mathtt{pk}_1^k = m \cdot Z^{ak}), 1)$;
otherwise return the (level-2) ciphertext $((e'(\mathtt{pk}_2, g_2)^k = Z^{bk}, m \cdot Z^k), 2)$.
5. $\mathsf{RE}(\mathtt{rk}_{i,j}, \mathtt{pk}_i, (\mathtt{c}_i, 1))$: Parse $\mathtt{c}_i$ as $(\mathtt{c}_{i,1}, \mathtt{c}_{i,2}) \in \mathbb{G}_2 \times \mathbb{G}_T$ and $\mathtt{pk}_i$ as $(\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2}) \in \mathbb{G}_T \times \mathbb{G}_1$. Pick $k' \leftarrow \mathbb{Z}_p$ for rerandomization and return the level-2 ciphertext $\boxed{((e'(\mathtt{rk}_{i,j}, \mathtt{c}_{i,1} \cdot g_2^{k'}), \mathtt{c}_{i,2} \cdot \mathtt{pk}_{i,1}^{k'}), 2)} = (Z^{b_j \cdot a_i (k+k')}, m \cdot Z^{a_i (k+k')})$.
6. $\mathsf{D}(\mathtt{sk}, (\mathtt{c}, l))$: Parse the secret key $\mathtt{sk}$ as (a, b). Parse a level-1 ciphertext as $c = (\mathtt{c}_1, \mathtt{c}_2) \in \mathbb{G}_2 \times \mathbb{G}_T$ and return $\mathtt{c}_2 / e'(g_1, \mathtt{c}_1)^a$. Parse a level-2 ciphertext as $c = (\mathtt{c}_1, \mathtt{c}_2) \in \mathbb{G}_T^2$ and return $\mathtt{c}_2 / c_1^{1/b}$.

Construction 2: Rerandomised [AFGH05] in the asymmetric setting. The differences from Construction 1 are highlighted in boxes.

Lemma 14. *If XDH is (t_2, ϵ_2)-hard then Construction 2 is $(t_2 - \delta_{out} \cdot t_{\mathsf{Exp}}, \epsilon_2, \delta_{out})$-weakly key-private, where t_{Exp} is the complexity of four exponentiations in $\mathbb{G}_1$.*

Proof. For $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e') \leftarrow \mathsf{BS}'(1^\lambda)$ and $Z := e'(g_1, g_2)$, our goal is to show that $\mathsf{KP}^0 \approx_{(t_2, \epsilon_2)} \mathsf{KP}^1$, where

$$\mathsf{KP}^0 := ((Z^{a_0}, g_1^{b_0}), (Z^{a_1}, g_1^{b_1}), \ldots, (Z^{a_{\delta_{out}}}, g_1^{b_{\delta_{out}}}), g_1^{b_1 \cdot a_0}, \ldots, g_1^{b_{\delta_{out}} \cdot a_0}) \text{ and}$$
$$\mathsf{KP}^1 := ((Z^{a_0}, g_1^{b_0}), (Z^{a_1}, g_1^{b_1}), \ldots, (Z^{a_{\delta_{out}}}, g_1^{b_{\delta_{out}}}), g_1^{b_1 \cdot r_1}, \ldots, g_1^{b_{\delta_{out}} \cdot r_{\delta_{out}}})$$

with $a_0, b_0, a_1, b_1, \ldots, a_{\delta_{out}}, b_{\delta_{out}}, r_1, \ldots, r_{\delta_{out}} \leftarrow \mathbb{Z}_p$. Let A be an adversary of size t_2 that distinguishes KP^0 from KP^1 with probability at least ϵ_2. Given an XDH instance (A, B, C, Grp), where $Grp := (p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e')$, the reduction first uses random self-reducibility of XDH to generate δ_{out} instances $(A, B_i, \mathtt{c}_i, Grp)$: it picks $u_i, v_i \leftarrow \mathbb{Z}_p$ and sets $B_i := B^{u_i} g_1^{v_i}$ and $\mathtt{c}_i := C^{u_i} A^{v_i}$. (If $A = g_1^a, B = g_1^b, C = g_1^c$ then $B_i = g_1^{b_i}$ with $b_i = bu_i + v_i$ and $\mathtt{c}_i = g_1^{c_i}$ with $\mathtt{c}_i = u_i(c - ab) + ab_i$; thus if $c = ab$ then $(A, B_i, \mathtt{c}_i)$ is a DH tuple, otherwise B_i and $\mathtt{c}_i$ are independently random.) The reduction picks $b_0, a_1, \ldots, a_{\delta_{out}} \leftarrow \mathbb{Z}_p$ and sends

$$\mathsf{KP} := ((e'(A, g_2), g_1^{b_0}), (Z^{a_1}, B_1), \ldots, (Z^{a_{\delta_{out}}}, B_{\delta_{out}}), \mathtt{c}_1, \ldots, \mathtt{c}_{\delta_{out}})$$

to A. Depending on whether C was real or random, the adversary sees either KP^0 or KP^1, and any distinguishing advantage it has is translated to that of the reduction. □

Note that the reduction of XDH to δ_{out}-weak key-privacy is without any security loss due to the use of random self-reducibility of XDH. The adaptive security of Construction 2 against CPA (resp., HRA) is now follows from Theorem 7 (resp., Theorem 8), [AFGH05, Theorem 3.1], Lemma 13 and Lemma 14.

Theorem 9 (PRE-CPA and PRE-HRA security of Construction 2)**.** *Let σ and τ denote, respectively, the space and time complexity for the class $\mathcal{G} = \mathcal{G}(N, \delta_{out}, D)$. Assume* BS′ *generates asymmetric bilinear groups for which eDBDH is (t_1, ϵ_1)-hard, XDH is (t_2, ϵ_2)-hard and where four exponentiations in $\mathbb{G}_1$ cost t_{Exp}. Then Construction 2 is* (t, ϵ)-PRE-CPA-*secure and* (t', ϵ')-PRE-HRA-*secure restricted to challenge graphs in $\mathcal{G}$ where*

$$
\begin{aligned}
t &:= \min(t_1, t_2 - \delta_{out} \cdot t_{\mathsf{Exp}}) - t_{\mathsf{CPA}} - t_{\mathcal{G}},\\
\epsilon &:= 2(\epsilon_1 + \tau \cdot \epsilon_2) \cdot N^{\sigma+\delta_{out}+1},\\
t' &:= \min(t_1, t_2 - \delta_{out} \cdot t_{\mathsf{Exp}}) - t_{\mathsf{HRA}} - t_{\mathcal{G}} \quad \textit{and}\\
\epsilon' &:= 2N(N-1)(Q_{\mathsf{E}} + Q_{\mathsf{RE}})Q_{\mathsf{RE}} \cdot 2^{-\lambda} + \epsilon.
\end{aligned}
$$

The ABH Scheme.

This scheme, given in Construction 3, can be thought of as a variant of Construction 1 with a *randomized* re-key generation algorithm, and it is this feature that enabled [ABH09] to show (strong) key-privacy assuming DLin. In Construction 4, we simplify their scheme and show that it is still *weakly* key-private assuming DLin. The main difference from the original construction is the way the re-randomization of a re-encrypted ciphertext is carried out: it is now done just like in Construction 2, and this allows for shorter re-keys (just two group elements compared to four).

Security. If the eDBDH problem is (t_1, ϵ_1)-hard then Construction 3 is $(t_1, 2\epsilon_1)$-sPRE-CPA-secure [ABH09, Theorem 3.1]. The sPRE-CPA security of Construction 4 follows by the same reduction: we refer the readers to [ABH09] for the details. It was also shown in [ABH09, Theorem 3.4] that if the DLin problem is (t_2, ϵ_2)-hard then Construction 3 is $(t_2, 4N^2 \cdot \epsilon_2)$-(strongly) key-private. Below we simplify this reduction to show that Construction 4 is $(t_2, \delta_{out} \cdot \epsilon_2, \delta_{out})$-weakly key-private (Lemma 15). We also show (Lemma 16) that it is statistically source-hiding.

Lemma 15. *If the DLin problem is (t_2, ϵ_2)-hard then Construction 4 is $(t_2, \delta_{out} \cdot \epsilon_2, \delta_{out})$-weakly key-private.*

Proof. For $(p, g, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathsf{BS}(1^\lambda)$, a random generator $h \in \mathbb{G}$ and $Z := e(g, h)$, our goal is to show that $\mathsf{KP}^0 \approx_{(t_2,\epsilon_2)} \mathsf{KP}^1$, where

$$
\begin{aligned}
\mathsf{KP}^0 &:= \Big((Z^{a_0}, g^{b_0}), \ldots, (Z^{a_{\delta_{out}}}, g^{b_{\delta_{out}}}), ((g^{b_1})^{a_0+r_1}, h^{r_1}), \ldots, ((g^{b_{\delta_{out}}})^{a_0+r_{\delta_{out}}}, h^{r_{\delta_{out}}})\Big) \quad \text{and}\\
\mathsf{KP}^1 &:= \Big((Z^{a_0}, g^{b_0}), \ldots, (Z^{a_{\delta_{out}}}, g^{b_{\delta_{out}}}), ((g^{b_1})^{a'_1+r_1}, h^{r_1}), \ldots, ((g^{b_{\delta_{out}}})^{a'_{\delta_{out}}+r_{\delta_{out}}}, h^{r_{\delta_{out}}})\Big)
\end{aligned}
$$

with $a_0, b_0, a_1, a'_1, b_1, r_1, \ldots, a_{\delta_{out}}, a'_{\delta_{out}}, b_{\delta_{out}}, r_{\delta_{out}} \leftarrow \mathbb{Z}_p$. Unfortunately, in contrast to Lemma 14, random self-reducibility of DLin does not apply here, since the DLin instances share one of their exponents (a_0); we therefore proceed via a hybrid argument. Let A be an adversary of size t_2 that distinguishes KP^0 from KP^1 with probability at least ϵ_2. With probability ϵ_2/δ_{out} it

1. $\mathsf{S}(1^\lambda)$: $(p, g, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathsf{BS}(1^\lambda)$. Pick a random generator $h \in \mathbb{G}$ and compute $Z = e(g, h)$. Return the public parameters $\mathtt{pp} = ((p, g, \mathbb{G}, \mathbb{G}_T, e), h, Z)$.

2. $\mathsf{K}(\mathtt{pp})$: Pick $a, b \leftarrow \mathbb{Z}_p$ and set the public key as $\mathtt{pk} := (Z^a, g^b)$ and the secret key as $\mathtt{sk} := (a, b)$. Return the keys $(\mathtt{pk}, \mathtt{sk})$.

3. $\mathsf{RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$: Parse $\mathtt{sk}_i$ as $(a_i, b_i) \in \mathbb{Z}_p^2$ and $\mathtt{pk}_j$ as $(\mathtt{pk}_{j,1}, \mathtt{pk}_{j,2}) \in \mathbb{G}_T \times \mathbb{G}$. Pick $r, \boxed{w} \leftarrow \mathbb{Z}_p$ and return the re-key:

$$\mathtt{rk}_{i,j} := ((\mathtt{pk}_{j,2})^{a_i + r}, h^r, \boxed{e(\mathtt{pk}_{j,2}, h)^w, e(g, h)^w}) = ((g^{b_j})^{a_i + r}, h^r, \boxed{Z^{wb_j}, Z^w}).$$

4. $\mathsf{E}(\mathtt{pk}, (\mathtt{m}, l))$: Parse the public key as $\mathtt{pk} = (\mathtt{pk}_1, \mathtt{pk}_2) \in \mathbb{G}_T \times \mathbb{G}$ and pick $k \leftarrow \mathbb{Z}_p$. If $l = 1$ return the (level-1) ciphertext $((g^k, h^k, m \cdot \mathtt{pk}_1^k), 1)$; otherwise, return the (level-2) ciphertext $((e(\mathtt{pk}_2, h)^k, \mathtt{m} \cdot Z^k), 2)$.

5. $\mathsf{RE}(\mathtt{rk}_{i,j}, (\mathtt{c}_i, 1))$: Parse the re-key $\mathtt{rk}_{i,j}$ as $(\mathtt{rk}_{i,j,1}, \ldots, \mathtt{rk}_{i,j,4}) \in \mathbb{G}^2 \times \mathbb{G}_T^2$, and the ciphertext $\mathtt{c}_i$ as $(\mathtt{c}_{i,1}, \mathtt{c}_{i,2}, \mathtt{c}_{i,3}) \in \mathbb{G}^2 \times \mathbb{G}_T$. Verify that the ciphertext is well-formed by checking if $e(\mathtt{c}_{i,1}, h) = e(g, \mathtt{c}_{i,2})$ – if it is not, halt. Compute $t_1 = e(\mathtt{rk}_{i,j,1}, \mathtt{c}_{i,2}) = Z^{b_j(a_i + r)k}$ and $t_2 = c_{i,3} \cdot e(\mathtt{c}_{i,1}, \mathtt{rk}_{i,j,2}) = \mathtt{m} \cdot Z^{a_i k} \cdot Z^{kr}$; choose $w' \leftarrow \mathbb{Z}_p$ and re-randomise t_1, t_2 by setting $t_1' = t_1 \cdot \boxed{\mathtt{rk}_{i,j,3}^{w'}}$ and $t_2' = t_2 \cdot \boxed{\mathtt{rk}_{i,j,4}^{w'}}$. Return $((t_1', t_2'), 2)$.

6. $\mathsf{D}(\mathtt{sk}, (\mathtt{c}, l))$: Parse the secret key $\mathtt{sk}$ as $(a, b) \in \mathbb{Z}_p^2$. Parse a level-1 ciphertext as $(\mathtt{c}_1, \mathtt{c}_2, \mathtt{c}_3) \in \mathbb{G}^2 \times \mathbb{G}_T$; halt if $e(\mathtt{c}_1, h) \neq e(g, \mathtt{c}_2)$, and otherwise return $\mathtt{c}_3 / e(\mathtt{c}_1, h)^a$. Parse a level-2 ciphertext as as $(\mathtt{c}_1, \mathtt{c}_2) \in \mathbb{G}_T^2$ and return $\mathtt{c}_2 / c_1^{1/b}$.

Construction 3: Unidirectional, single-hop PRE from [ABH09]: the differences to our simplified version (Construction 4) are boxed.

must therefore distinguish two of the hybrid games $\mathsf{KP}^0 = \mathsf{KP}_0, \ldots, \mathsf{KP}_{\delta_{out}} = \mathsf{KP}^1$ defined as

$$\mathsf{KP}_i := \Big((Z^{a_0}, g^{b_0}), \ldots, (Z^{a_{\delta_{out}}}, g^{b_{\delta_{out}}}), ((g^{b_1})^{a_1' + r_1}, h^{r_1}), \ldots, ((g^{b_i})^{a_i' + r_i}, h^{r_i}), \\ ((g^{b_{i+1}})^{a_0 + r_{i+1}}, h^{r_{i+1}}), \ldots ((g^{b_{\delta_{out}}})^{a_0 + r_{\delta_{out}}}, h^{r_{\delta_{out}}})\Big).$$

Given a DLin instance (h, f, A, B, C, Grp), where $Grp := (p, g, \mathbb{G}, \mathbb{G}_T, e)$, the reduction picks $a_1, \ldots, a_{\delta_{out}}, b_0, r_0, \ldots, b_{i-1}, r_{i-1}, b_{i+1}, r_{i+1}, \ldots, b_{\delta_{out}}, r_{\delta_{out}} \leftarrow \mathbb{Z}_p$ and runs A on

$$\Big((e(A, h), g^{b_0}), (Z^{a_1}, g^{b_1}), \ldots, (Z^{a_{i-1}}, g^{b_{i-1}}), (Z^{a_i}, f), (Z^{a_{i+1}}, g^{b_{i+1}}), \ldots, (Z^{a_{\delta_{out}}}, g^{b_{\delta_{out}}}), \\ ((g^{b_1})^{a_1' + r_1}, h^{r_1}), \ldots, ((g^{b_{i-1}})^{a_{i-1}' + r_{i-1}}, h^{r_{i-1}}), (\mathtt{c}, B), \\ (A^{b_{i+1}} g^{b_{i+1} r_{i+1}}, h^{r_{i+1}}), \ldots, (A^{b_{\delta_{out}}} g^{b_{\delta_{out}} r_{\delta_{out}}}, h^{r_{\delta_{out}}})\Big).$$

Letting a_0, r_i be such that $A = g^{a_0}$ and $B = h^{r_i}$, the challenge C is either $f^{a_0 + r_i}$ (real) or $f^{a_i' + r_i}$ (random). Thus the above is either distributed as KP_{i-1} (real) or KP_i (random). Its distinguishing advantage thus translates to that of the reduction. □

1. $\mathsf{S}(1^\lambda)$: $(p, g, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathsf{BS}(1^\lambda)$. Pick a random generator $h \in \mathbb{G}$ and compute $Z = e(g, h)$. Return the public parameters $\mathtt{pp} = ((p, g, \mathbb{G}, \mathbb{G}_T, e), h, Z)$.

2. $\mathsf{K}(\mathtt{pp})$: Pick $a, b \leftarrow \mathbb{Z}_p$ and set the public key as $\mathtt{pk} := (Z^a, g^b)$ and the secret key as $\mathtt{sk} := (a, b)$. Return the keys $(\mathtt{pk}, \mathtt{sk})$.

3. $\mathsf{RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$: Parse $\mathtt{sk}_i$ as $(a_i, b_i) \in \mathbb{Z}_p^2$ and $\mathtt{pk}_j$ as $(\mathtt{pk}_{j,1}, \mathtt{pk}_{j,2}) \in \mathbb{G}_T \times \mathbb{G}$. Pick $r \leftarrow \mathbb{Z}_p$ and return the re-key:

$$\mathtt{rk}_{i,j} := ((\mathtt{pk}_{j,2})^{a_i + r}, h^r) = ((g^{b_j})^{a_i + r}, h^r).$$

4. $\mathsf{E}(\mathtt{pk}, (\mathtt{m}, l))$: Parse the public key as $\mathtt{pk} = (\mathtt{pk}_1, \mathtt{pk}_2) \in \mathbb{G}_T \times \mathbb{G}$ and pick $k \leftarrow \mathbb{Z}_p$. If $l = 1$ return the (level-1) ciphertext $(\mathtt{c}, 1) = ((g^k, h^k, \mathtt{m} \cdot \mathtt{pk}_1^k), 1)$; otherwise, return the (level-2) ciphertext $(\mathtt{c}, 2) = ((e(\mathtt{pk}_2, h)^k = Z^{b \cdot k}, \mathtt{m} \cdot Z^k), 2)$.

5. $\mathsf{RE}(\mathtt{rk}_{i,j}, \mathtt{pk}_i, (\mathtt{c}_i, 1))$: Parse the re-key $\mathtt{rk}_{i,j}$ as $(\mathtt{rk}_{i,j,1}, \mathtt{rk}_{i,j,2}) \in \mathbb{G}^2$, the public key $\mathtt{pk}_i$ as $(\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2}) \in \mathbb{G}_T \times \mathbb{G}$ and the ciphertext $\mathtt{c}_i$ as $(\mathtt{c}_{i,1}, \mathtt{c}_{i,2}, \mathtt{c}_{i,3}) \in \mathbb{G}^2 \times \mathbb{G}_T$. Verify that the ciphertext is well-formed by checking if $e(\mathtt{c}_{i,1}, h) = e(g, \mathtt{c}_{i,2})$ – if it is not, halt. Pick $k' \leftarrow \mathbb{Z}_p$ and return $((\mathtt{c}_{j,1}, \mathtt{c}_{j,2}), 2)$, where

$$c_{j,1} := e(\mathtt{rk}_{i,j,1},\ \mathtt{c}_{i,2} \cdot h^{k'}) = Z^{b_j(a_i + r)(k + k')} \quad \text{and}$$
$$c_{j,2} := c_{i,3} \cdot \mathtt{pk}_{i,1}^{k'} \cdot e(\mathtt{c}_{i,1} \cdot g^{k'},\ \mathtt{rk}_{i,j,2}) = m \cdot Z^{a_i k} \cdot Z^{a_i k'} \cdot Z^{(k + k')r}.$$

6. $\mathsf{D}(\mathtt{sk}, (\mathtt{c}, l))$: Parse the secret key $\mathtt{sk}$ as $(a, b) \in \mathbb{Z}_p$. Parse a level-1 ciphertext as $(\mathtt{c}_1, \mathtt{c}_2, \mathtt{c}_3) \in \mathbb{G}^2 \times \mathbb{G}_T$; halt if $e(\mathtt{c}_1, h) \neq e(g, \mathtt{c}_2)$, and otherwise return $\mathtt{c}_3 / e(\mathtt{c}_1, h)^a$. Parse a level-2 ciphertext as $(\mathtt{c}_1, \mathtt{c}_2) \in \mathbb{G}_T^2$ and return $\mathtt{c}_2 / c_1^{1/b}$.

Construction 4: Simplified version of scheme from [ABH09] (Construction 3).

Lemma 16. *Construction 4 is statistically source-hiding.*

Proof. A level-2 ciphertext under $\mathtt{pk}_j$ that results from the re-encryption of a level-1 ciphertext $\mathtt{c}_i = (\mathtt{c}_{i,1}, \mathtt{c}_{i,2}, \mathtt{c}_{i,3})$ under $\mathtt{pk}_i = (\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2})$ is of the form

$$\begin{aligned} c_j &= (\mathtt{c}_{j,1}, \mathtt{c}_{j,2}) \\ &= (e(\mathtt{rk}_{i,j,1}, \mathtt{c}_{i,2} \cdot h^{k'}), \mathtt{c}_{i,3} \cdot \mathtt{pk}_{i,1}^{k'} \cdot e(\mathtt{c}_{i,1} \cdot g^{k'}, \mathtt{rk}_{i,j,2})) \\ &= (e(g^{b_j(a_i + r)}, h^k \cdot h^{k'}), \mathtt{m} \cdot Z^{a_i k} \cdot Z^{a_i k'} \cdot e(g^k \cdot g^{k'}, h^r)) \\ &= (Z^{b_j(k + k')(a_i + r)}, \mathtt{m} \cdot Z^{(k + k')(a_i + r)}). \end{aligned} \tag{4.3}$$

From (4.3) it is clear that the distribution of the re-encrypted ciphertext is statistically close to a fresh level-2 ciphertext under $\mathtt{pk}_j$. □

The adaptive security of Construction 4 against CPA (resp., HRA) now follows from Theorem 7 (resp., Theorem 8) and [ABH09, Theorem 3.1] using the above lemmas.

Theorem 10 (PRE-CPA and PRE-HRA security of Construction 4)**.** *Let σ and τ denote, resp., the space and time complexity for the class $\mathcal{G} = \mathcal{G}(N, \delta_{out}, D)$. Assume* BS *generates bilinear groups for which eDBDH is (t_1, ϵ_1)-hard and DLin is (t_2, ϵ_2)-hard. Then Construction 4 is (t, ϵ)-*PRE-CPA*-secure and (t', ϵ')-*PRE-HRA*-secure restricted to challenge graphs in $\mathcal{G}$ where*

$$t := \min(t_1, t_2) - t_{\mathsf{CPA}} - t_{\mathcal{G}} \qquad \epsilon := 2(\epsilon_1 + \tau \cdot \delta_{out} \cdot \epsilon_2) \cdot N^{\sigma+\delta_{out}+1}$$
$$t' := \min(t_1, t_2) - t_{\mathsf{HRA}} - t_{\mathcal{G}} \qquad \epsilon' := 2N(N-1)(Q_{\mathsf{E}} + Q_{\mathsf{RE}})Q_{\mathsf{RE}} \cdot 2^{-\lambda} + \epsilon$$

Remark 2. We note that since the proofs for key-privacy in both Constructions 2 and 4 proceed via a hybrid argument, by using a trick of Panjwani [Pan07], one can improve the bound for ϵ to $2(\epsilon_1 + \tau \cdot \epsilon_2) \cdot N^{\sigma+\log\delta_{out}+1}$ and $2(\epsilon_1 + \tau \cdot \delta_{out} \cdot \epsilon_2) \cdot N^{\sigma+\log\delta_{out}+1}$, respectively. We refer the reader to Chapter 3.

4.5.2 Multi-Hop Scheme from Fully Homomorphic Encryption.

We now describe the generic construction of a unidirectional multi-hop PRE scheme from fully homomorphic encryption (FHE) due to Gentry [Gen09].

Gentry's Scheme.

A fully homomorphic encryption scheme consists of a five-tuple of algorithms $(\mathsf{S}, \mathsf{K}, \mathsf{E}, \mathsf{D}, \mathsf{F})$, where S is the setup algorithm, K the key-generation algorithm, E the encryption algorithm, D the decryption algorithm and F the homomorphic evaluation algorithm, which takes a function f and encrypted inputs for f and returns an encryption of the evaluation of f on the inputs.

Gentry [Gen09] gave a generic construction of PRE from FHE. We show that this scheme, given in Construction 5.a, is adaptively secure against CPA. In Construction 5.b, we "sanitize" the previous construction and prove that the resulting construction is adaptively secure against HRA.

Security. The ciphertext indistinguishability of Construction 5.a directly follows from the semantic security of the underlying FHE scheme. We show that weak key-privacy also follows from semantic security. If, in addition, the FHE scheme is *sanitizable*, then Construction 5.b is source-hiding and thus satisfies PRE-HRA. In particular, if sanitizability is as defined and discussed in [DS16] (see Definition 32), then Construction 5.b is *statistically* source-hiding.

Lemma 17. *If* FHE *is (t_1, ϵ_1)-semantically secure then Construction 5.a is $(t_1 - \delta_{out} \cdot t_{\mathsf{E}} - N \cdot t_{\mathsf{K}}, \delta_{out} \cdot \epsilon_1, \delta_{out})$-weakly key-private.*

Proof. Our goal is to show that $\mathsf{KP}^0 \approx_{(t_1,\epsilon_1)} \mathsf{KP}^1$ where

$$\mathsf{KP}^0 := (\mathrm{pp}, \mathrm{pk}_0, \mathrm{pk}_1, \ldots, \mathrm{pk}_{\delta_{out}}, \mathsf{FHE.E}(\mathrm{pk}_1, \mathrm{sk}_0), \ldots, \mathsf{FHE.E}(\mathrm{pk}_{\delta_{out}}, \mathrm{sk}_0)) \text{ and}$$
$$\mathsf{KP}^1 := (\mathrm{pp}, \mathrm{pk}_0, \mathrm{pk}_1, \ldots, \mathrm{pk}_{\delta_{out}}, \mathsf{FHE.E}(\mathrm{pk}_1, \mathrm{sk}'_0), \ldots, \mathsf{FHE.E}(\mathrm{pk}_{\delta_{out}}, \mathrm{sk}'_{\delta_{out}})),$$

where $\mathrm{pp} \leftarrow \mathsf{FHE.S}(1^\lambda)$ and $(\mathrm{pk}_0, \mathrm{sk}_0), \ldots, (\mathrm{pk}_{\delta_{out}}, \mathrm{sk}_{\delta_{out}}), (\mathrm{pk}'_1, \mathrm{sk}'_1), \ldots, (\mathrm{pk}'_{\delta_{out}}, \mathrm{sk}'_{\delta_{out}}) \leftarrow \mathsf{FHE.K}(\mathrm{pp})$. We use a sequence of hybrid distributions $\mathsf{KP}^0 = \mathsf{KP}_0, \ldots, \mathsf{KP}_{\delta_{out}} = \mathsf{KP}^1$, where in the i-th hybrid the first i re-keys are random and the rest real. That is,

$$\mathsf{KP}_i := \Big(\mathrm{pp}, \mathrm{pk}_0, \mathrm{pk}_1, \ldots, \mathrm{pk}_{\delta_{out}}, \mathsf{FHE.E}(\mathrm{pk}_1, \mathrm{sk}'_0), \ldots, \mathsf{FHE.E}(\mathrm{pk}_i, \mathrm{sk}'_i), \mathsf{FHE.E}(\mathrm{pk}_{i+1}, \mathrm{sk}_0), \ldots, \mathsf{FHE.E}(\mathrm{pk}_{\delta_{out}}, \mathrm{sk}_0)\Big).$$

1. Algorithms S,K and D for PRE are defined the same as their counterparts in FHE.
2. PRE.RK$((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$: The re-key $\mathtt{rk}_{i,j}$ is an encryption of $\mathtt{sk}_i$ under $\mathtt{pk}_j$:
$$\mathsf{PRE.RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j) := \mathsf{FHE.E}(\mathtt{pk}_j, \mathtt{sk}_i).$$
3. PRE.E$(\mathtt{pk}, \mathtt{m})$: Given a message $\mathtt{m}$ and a public key $\mathtt{pk}$, the encryption algorithm outputs a "sanitized" FHE encryption:
$$(\mathsf{PRE.E}(\mathtt{pk}, \mathtt{m}) := \boxed{\mathsf{Sanitize}\ (\mathtt{pk},}\ \mathsf{FHE.E}(\mathtt{pk}, \mathtt{m})).$$
4. PRE.RE$(\mathtt{rk}_{i,j}, \mathtt{pk}_i, \mathtt{pk}_j, (\mathtt{c}_i, l)) \to (\mathtt{c}_j, l+1)$: Given a re-key $\mathtt{rk}_{i,j}$ and a level-l ciphertext $\mathtt{c}_i$ that was encrypted under $\mathtt{pk}_i$, the re-encryption algorithm homomorphically decrypts the ciphertext $\mathtt{c}_i$ and "sanitizes" the result:
$$c_j \leftarrow \boxed{\mathsf{Sanitize}(\mathtt{pk}_j,}\ \mathsf{FHE.F}(\mathsf{FHE.D}, (\mathtt{rk}_{i,j}, \mathsf{FHE.E}(\mathtt{pk}_j, \mathtt{c}_i)))).$$

Construction 5: PRE from sanitizable FHE. Sanitize denotes the sanitization algorithm. We refer to the construction without the boxes by Construction 5.a and the construction with blurring (including the boxes) by Construction 5.b.

Let A be an adversary of size t_1 that distinguishes KP^0 from KP^1 with probability at least ϵ_1. Given a challenge for FHE semantic security containing parameters $\mathtt{pp}^*$ and public key $\mathtt{pk}^*$, the reduction sets $\mathtt{pk}_i := \mathtt{pk}^*$, picks $(\mathtt{pk}_0, \mathtt{sk}_0), (\mathtt{pk}'_i, \mathtt{sk}'_i) \leftarrow \mathsf{FHE.K}(\mathtt{pp}^*)$, sends $(\mathtt{sk}_0, \mathtt{sk}'_i)$ to its own challenger and receives $\mathtt{c}^*$. The reduction embeds $\mathtt{c}^*$ at position i and sends

$$\mathsf{KP}_{i-1,i} := \Big(\mathtt{pp}, \mathtt{pk}_0, \mathtt{pk}_1, \ldots, \mathtt{pk}_{\delta_{out}}, \mathsf{FHE.E}(\mathtt{pk}_1, \mathtt{sk}'_0), \ldots, \mathsf{FHE.E}(\mathtt{pk}_{i-1}, \mathtt{sk}'_{i-1}), \mathtt{c}^*, \mathsf{FHE.E}(\mathtt{pk}_{i+1}, \mathtt{sk}_0), \ldots, \mathsf{FHE.E}(\mathtt{pk}_{\delta_{out}}, \mathtt{sk}_0)\Big)$$

to A. Depending on whether $\mathtt{c}^*$ encrypts $\mathtt{sk}_0$ or $\mathtt{sk}'_i$, $\mathsf{KP}_{i-1,i}$ is distributed as KP_{i-1} or KP_i. □

PRE-CPA security of Construction 5.a now follows from Theorem 7.

Theorem 11 (PRE-CPA security of Construction 5.a)**.** *Let σ and τ denote the space and time complexity for $\mathcal{G} = \mathcal{G}(N, \delta_{out}, D)$. If* FHE *is (t_1, ϵ_1)-semantically secure then Construction 5.a is* (t, ϵ)-PRE-CPA*-secure restricted to challenge graphs in $\mathcal{G}$, where*

$$t := t_1 - (\delta_{out} \cdot t_{\mathsf{E}} + N \cdot t_{\mathsf{K}} + t_{\mathsf{CPA}} + t_{\mathcal{G}}) \quad \textit{and} \quad \epsilon := (2\tau \cdot \delta_{out} + 1) \cdot \epsilon_1 \cdot N^{\sigma + \delta_{out} + 1}.$$

Definition 32 (Sanitizability of encryptions [DS16])**.** An encryption scheme $(\mathsf{S}, \mathsf{K}, \mathsf{E}, \mathsf{D})$ is called sanitizable if there exists a polynomial-time algorithm Sanitize which takes as input a public key and a ciphertext, outputs a ciphertext, and satisfies the following two properties (with all but negligible probability):

- *Message-preserving.* For any key pair $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{K}(1^\lambda)$ and any c in the ciphertext space
$$\mathsf{D}(\mathtt{sk}, \mathsf{Sanitize}(\mathtt{pk}, \mathtt{c})) = \mathsf{D}(\mathtt{sk}, \mathtt{c}).$$
- *Sanitizing.* For all $\mathtt{c}, \mathtt{c}'$ such that $\mathsf{D}(\mathtt{sk}, \mathtt{c}) = \mathsf{D}(\mathtt{sk}, \mathtt{c}')$
$$\Delta((\mathsf{Sanitize}(\mathtt{pk}, \mathtt{c}), \mathtt{pk}, \mathtt{sk}), (\mathsf{Sanitize}(\mathtt{pk}, \mathtt{c}'), \mathtt{pk}, \mathtt{sk})) \leq 2^{-\lambda}.$$

PRE-HRA security of Construction 5.b easily follows from Theorem 7.

Theorem 12 (PRE-HRA-security of Construction 5.b)**.** *Let ϵ be defined as in Theorem 11 and let σ and τ denote the space and time complexity for $\mathcal{G} = \mathcal{G}(N, \delta_{out}, D)$. If* FHE *is a sanitizable FHE scheme that is (t_1, ϵ_1)-semantically secure, then Construction 5.b is (t', ϵ')-*PRE-HRA*-secure restricted to challenge graphs in $\mathcal{G}$, where*

$$s' := t_1 - (\delta_{out} \cdot t_{\mathsf{E}} + N \cdot t_{\mathsf{K}} + t_{\mathsf{HRA}} + t_{\mathcal{G}}) \quad \textit{and} \quad \epsilon' := N(N-1)(Q_{\mathsf{E}} + Q_{\mathsf{RE}})Q_{\mathsf{RE}} \cdot 2^{-\lambda} + \epsilon.$$

4.5.3 Lattice-based Multi-Hop Schemes

Here, we decribe the lattice-based unidirectional multi-hop PRE scheme from [CCL+14]. Being based directly on the decision LWE (DLWE) problem it achieves better parameters than the construction from FHE above.

The CCL+ Scheme.

In [CCL+14], Chandran et al. propose two lattice-based unidirectional multi-hop PRE schemes. The schemes are built upon Regev's encryption [Reg05] and its dual version [GPV07], respectively. Here, we will describe the former one, which is inspired by the fully homomorphic encryption scheme of [BV11]. Security can be proven assuming the hardness of the decisional learning with errors (DLWE) problem (cf. Definition 33 below).

We recall Regev's encryption scheme RGV in Construction 6. We can now define the PRE scheme from [CCL+14] using RGV in Construction 7.a. To achieve source-hiding, Chandran et al. propose the variant given as Construction 7.b.

In both schemes, the LWE error will grow with each re-encryption and the level bound L needs to be chosen appropriately so that correctness of decryption is still guaranteed (with overwhelming probability). The second variant achieves the stronger notion of PRE-HRA-security (see below) at the cost of worse parameters; only a small number L of re-encryptions is supported by this scheme and the underlying security assumption is very strong.

Security. We will first show PRE-CPA-security of Construction 7.a and then consider PRE-HRA-security of Construction 7.b.

Definition 33 (DLWE [Reg05])**.** Let M, M', $p \in \mathbb{N}$. For a matrix $\boldsymbol{A} \leftarrow \mathbb{Z}_p^{M' \times M}$ and a secret vector $\boldsymbol{s} \leftarrow \mathbb{Z}_p^M$, each sampled uniformly at random, and a vector $\boldsymbol{e} \leftarrow \chi$ for an error distribution χ on $\mathbb{Z}_p^{M'}$, the decisional LWE problem $\mathrm{DLWE}_{M,M',p,\chi}$ is to distinguish $(\boldsymbol{A}, \boldsymbol{A} \cdot \boldsymbol{s} + \boldsymbol{e})$ from $(\boldsymbol{A}, \boldsymbol{b})$ for a uniformly random sample $\boldsymbol{b} \leftarrow \mathbb{Z}_p^{M'}$.

To prove adaptive security for the two variants of Construction 7, we will need the following lemma [BV11].

1. S(1^λ): Pick lattice parameters M, M', $p \in \mathbb{N}$ and a B-bounded error distribution χ on $\mathbb{Z}_p^{M'}$. Sample $\boldsymbol{A} \leftarrow \mathbb{Z}_p^{M' \times M}$ uniformly at random and return the public parameters $\mathtt{pp} = (\boldsymbol{A}, M, M', p, \chi)$.
2. K($\mathtt{pp}$): Sample $\boldsymbol{s} \leftarrow \mathbb{Z}_p^M$ uniformly at random and compute $\boldsymbol{b} = \boldsymbol{A} \cdot \boldsymbol{s} + \boldsymbol{e}$, where $\boldsymbol{e} \leftarrow \chi$. Set $\mathtt{pk} := \boldsymbol{b}$ as the public key and $\mathtt{sk} := \boldsymbol{s}$ as the secret key. Return $(\mathtt{pk}, \mathtt{sk})$.
3. E($\mathtt{pk}, \mathtt{m}$): On input $\mathtt{pk} \in \mathbb{Z}_p^{M'}$ and a message bit $\mathtt{m} \in \{0,1\}$, sample $\boldsymbol{r} \leftarrow \{0,1\}^{M'}$ and output
$$\mathbf{c} := \boldsymbol{r}^T(\boldsymbol{A}, \boldsymbol{b}) + (0^M, \mathtt{m} \cdot \lceil p/2 \rfloor) \in \mathbb{Z}_p^{M+1}.$$
4. D($\mathtt{sk}, \mathbf{c}$): On input a secret key $\mathtt{sk} = \boldsymbol{s} \in \mathbb{Z}_p^M$ and a ciphertext $\mathbf{c} = (\boldsymbol{\alpha}, \beta) \in \mathbb{Z}_p^M \times \mathbb{Z}_p$, output 0 if $\beta - \langle \boldsymbol{\alpha}, \boldsymbol{s} \rangle$ is closer to 0 than to $\lceil p/2 \rfloor$, else output 1.

Construction 6: Regev's encryption scheme RGV [Reg05].

Lemma 18 (Matrix-vector leftover hash lemma)**.** *Let λ, m, $p \in \mathbb{N}$, and $M' \geq M \cdot \log p + 2\lambda$. For $\boldsymbol{A} \leftarrow \mathbb{Z}_p^{M' \times M}$, $\boldsymbol{r} \leftarrow \{0,1\}^{M'}$, and $\boldsymbol{y} \leftarrow \mathbb{Z}_p^M$ each sampled uniformly at random, it holds $\Delta((\boldsymbol{A}, \boldsymbol{A}^T\boldsymbol{r}), (\boldsymbol{A}, \boldsymbol{y})) \leq 2^{-\lambda}$.*

Assuming the computational hardness of $\mathrm{DLWE}_{M,M',p,\chi}$ for appropriate parameters, we get for any $\mathtt{pp} = (\boldsymbol{A}, \mathtt{m}, M', p, \chi)$, $\mathtt{pk} = \boldsymbol{b}$ and $\mathtt{m} \in \{0,1\}$: RGV.E($\mathtt{pk}, \mathtt{m}$) $= \boldsymbol{r}^T(\boldsymbol{A}, \boldsymbol{b}) + (0^M, \mathtt{m} \cdot \lceil p/2 \rfloor)$ is computationally indistinguishable from $\boldsymbol{r}^T(\boldsymbol{A}, \boldsymbol{b}') + (0^M, \mathtt{m} \cdot \lceil p/2 \rfloor)$, where $\boldsymbol{r} \leftarrow \{0,1\}^{M'}$ and $\boldsymbol{b}' \leftarrow \mathbb{Z}_p^{M'}$. By the above lemma, the latter distribution is, in turn, statistically close to the uniform distribution on $\mathbb{Z}_p^{M+1}$. Informally, since RGV.E($\mathtt{pk}, 0$) is computationally indistinguishable from uniformly random, ciphertexts, re-keys and re-encrypted ciphertexts all look uniformly random; in particular Construction 7.a satisfies indistinguishability of ciphertexts as well as δ_{out}-weak key privacy.

Lemma 19. *Assuming $\mathrm{DLWE}_{M,M',p,\chi}$ is (t_1, ϵ_1)-hard for parameters M, M', p as in Lemma 18, Construction 7.a satisfies $(t_1 - t_{\mathsf{E}}, 2(\epsilon_1 + 2^{-\lambda}))$-indistinguishability and $(t_1 - O(\delta_{out}\, M \lceil \log p \rceil \cdot (t_{\mathbb{Z}_p^{M+1}} + t_{\mathsf{RGV.E}})), \delta_{out} M \lceil \log p \rceil \cdot \epsilon_1, \delta_{out})$-weak key-privacy.*

PRE-CPA security of Construction 7.a now follows from Theorem 7.

Theorem 13 (PRE-CPA-security of Construction 7.a)**.** *Let σ and τ denote the space and time complexity for the class $\mathcal{G} = \mathcal{G}(N, \delta_{out}, D)$. Assume the $\mathrm{DLWE}_{M,M',p,\chi}$ problem is (t_1, ϵ_1)-hard for parameters M, M', p as in Lemma 18. Then Construction 7.a is (t, ϵ)-*PRE-CPA*-secure restricted to challenge graphs in $\mathcal{G}$, where*

$$t := t_1 - O(\delta_{out}\, M \lceil \log p \rceil \cdot (t_{\mathbb{Z}_p^{M+1}} + t_{\mathsf{RGV.E}})) - t_{\mathsf{CPA}} - t_{\mathcal{G}} \quad \textit{and}$$
$$\epsilon := (2\tau \cdot \delta_{out} M \lceil \log p \rceil + 1) \cdot \epsilon_1 \cdot N^{\sigma + \delta_{out} + 1}.$$

1. $\mathsf{S}(1^\lambda)$: Get parameters $\mathtt{pp}' \leftarrow \mathsf{RGV.S}(1^\lambda)$, level bound L and "blurring error" bound E_l for each level $l \in [1, L]$. Return the parameters $\mathtt{pp} = (\mathtt{pp}', L, (E_l)_{l\in[1,L]})$.
2. $\mathsf{K}(\mathtt{pp})$: Run $\mathsf{RGV.K}(\mathtt{pp}')$ and output the result.
3. $\mathsf{E}(\mathtt{pk}, (\mathtt{m}, l))$: Compute the ciphertext
$$\boldsymbol{c} = \mathsf{RGV.E}(\mathtt{pk}, \mathtt{m}) + (0^M, f_l),$$
where $f_l \leftarrow [-E_l, E_l] \cap \mathbb{Z}$, and return the level-$l$ ciphertext $(\boldsymbol{c}, l)$.
4. $\mathsf{RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$: Parse $\mathtt{sk}_i$ as $\boldsymbol{s}_i = (t_{i,1}, \ldots, t_{i,M}) \in \mathbb{Z}_p^M$. For $k \in [1, M]$ and $l \in [1, \lceil \log p \rceil]$, compute $K_{k,l} \leftarrow \mathsf{RGV.E}(\mathtt{pk}_j, 0) + (0^M, t_{i,k} \cdot 2^l)$. Return the re-key:
$$\mathtt{rk}_{i,j} := \{K_{k,l}\}_{k\in[1,M], l\in[1,\lceil \log p\rceil]}.$$
5. $\mathsf{RE}(\mathtt{rk}_{i,j}, \mathtt{pk}_i, \mathtt{pk}_j, (\boldsymbol{c}_i, \ell)) \to (\boldsymbol{c}_j, \ell + 1)$: If $\ell \geq L$, abort. Otherwise, parse the level-l ciphertext $\boldsymbol{c}_i$ as $(\boldsymbol{\alpha}, \beta) \in \mathbb{Z}_p^M \times \mathbb{Z}_p$ and $\mathtt{rk}_{i,j}$ as $\{K_{k,l}\}_{k\in[1,M], l\in[1,\lceil \log p\rceil]}$. Denote by α_k the k-th component of $\boldsymbol{\alpha}$, and denote the bit decomposition of α_k as $\{\alpha_{k,l}\}_{l\in[1,\lceil \log p\rceil]}$, i.e., $\alpha_k = \sum_{l\in[1,\lceil \log p\rceil]} \alpha_{k,l} 2^l$, where each $\alpha_{k,l} \in \{0, 1\}$. Compute
$$\boldsymbol{c}_j = (0^M, \beta) + \textstyle\sum_{k,l} \alpha_{k,l} \cdot K_{k,l} + \mathsf{RGV.E}(\mathtt{pk}_j, 0) + (0^M, f_{l+1}),$$
where $f_{l+1} \leftarrow [-E_{l+1}, E_{l+1}] \cap \mathbb{Z}$, and return $(\boldsymbol{c}_j, \ell + 1)$.
6. $\mathsf{D}(\mathtt{sk}, (\boldsymbol{c}, \ell))$: Run $\mathsf{RGV.D}(\mathtt{sk}, \boldsymbol{c})$ and output the result.

Construction 7: source-hiding unidirectional multi-hop PRE from [CCL+14]. We refer to the construction without the blurring (ignoring the boxes) by Construction 7.a and the construction with blurring (including the boxes) by Construction 7.b.

Construction 7.a clearly does not satisfy source-hiding and, thus, cannot be proven PRE-HRA-secure using our results. Fortunately, Construction 7.b solves this issue, but at the cost of only allowing for a constant level bound L. The additional uniform error $f_l \leftarrow [-E_l, E_l] \cap \mathbb{Z}$ added in E and RE in Construction 7.b is used to "blur out" the different errors caused by encryption or re-encryption, respectively. Choosing the error bounds E_l appropriately guarantees the source-hiding property of the scheme while still preserving correctness.[6] Chandran et al. refer to this rerandomisation technique as *strong blurring*; a more detailed analysis can be found in [DS16, Section 4.1], where the same method for rerandomization of Regev ciphertexts is used

[6] In fact, we need to choose the error bounds $(E_l)_{l\in[1,L]}$ exponentially large, eg., $E_1 \geq (M' + 1)B2^\lambda$. Thus, to provide correctness of the scheme, one needs to choose the modulus p to be of size $\exp(O(\lambda))$ and the level bound L of size $O(1)$.

to discuss sanitizability of the FHE scheme from [BV11].

To prove PRE-HRA-security of Construction 7.b, note that, as above, semantic security and δ_{out}-weak key-privacy of (E, D) directly follow by the security of Regev's encryption scheme. We state the result for source-hiding only informally:.

Lemma 20. *For large enough (see Footnote 6) error ranges E_l, $l \in [1, L]$, Construction 7.b is (statistically) source-hiding.*

PRE-HRA security of Construction 7.a now follows from Theorem 8.

Theorem 14 (PRE-HRA-security of Construction 7.b)**.** *Let ϵ be as in Theorem 13 and let σ and τ denote the space and time complexity for $\mathcal{G} = \mathcal{G}(N, \delta_{out}, D)$. If $\mathrm{DLWE}_{M,M',p,\chi}$ is (t_1, ϵ_1)-hard for parameters M, M', p as in Lemma 18 and E_l ($l \in [1, L]$), L are chosen appropriately, then Construction 7.b is (t', ϵ')-*PRE-HRA*-secure restricted to challenge graphs in $\mathcal{G}$, where*

$$t' := t_1 - O(\delta_{out}\, M \lceil \log p \rceil \cdot (t_{\mathbb{Z}_p^{M+1}} + t_{\mathsf{RGV.E}})) - t_{\mathsf{HRA}} - t_{\mathcal{G}}, \quad \textit{and}$$
$$\epsilon' := 2N(N-1)(Q_{\mathsf{E}} + Q_{\mathsf{RE}})Q_{\mathsf{RE}} \cdot 2^{-\lambda} + \epsilon.$$

4.6 Application to Key Rotation

An interesting application of unidirectional multi-hop PRE is key rotation in remote storage systems, for which it can mitigate the risk of key compromise. Consider a user that stores content on a server which is encrypted under her public key $\mathtt{pk}$. Using key rotation, from time to time the user creates a new key pair $(\mathtt{pk}', \mathtt{sk}')$ and would like to replace the encryption of the content under $\mathtt{pk}$ by an encryption under $\mathtt{pk}'$.

To do so, it seems she must either download the content, re-encrypt it and upload it again; or give the secret key $\mathtt{sk}$ for $\mathtt{pk}$ to the server, so the latter can obtain the content and encrypt it under the new key. PRE allows avoiding both costly transfer of content *and* unnecessary trust in the server: the user simply creates $\mathtt{rk} \leftarrow \mathsf{RK}((\mathtt{pk}, \mathtt{sk}), \mathtt{pk}')$ and sends $\mathtt{rk}$ to the server. The latter uses $\mathtt{rk}$ to re-encrypt the content to the new key $\mathtt{pk}'$ without knowing either content nor any secret keys.

There are several attack scenarios to consider: an adversary could (i) obtain the encrypted content or (ii) a re-key by breaking into the server; and it could obtain (iii) the secret key by attacking the user. We would like to guarantee that, as long as the adversary does not see a ciphertext which it can decrypt by either obtaining the secret key for it or re-keys that allow it to (consecutively) re-encrypt it to a key it knows, nothing is leaked on the plaintext.

We formalize this via a game that considers N epochs in each of which a new key pair is generated and the content is re-encrypted. The adversary obtains all public keys and can ask for messages to be encrypted at certain epochs; one of which is its challenge query which lets it choose two messages $(\mathtt{m}_0, \mathtt{m}_1)$. The adversary can at any time query secret keys for any epoch and re-keys between two epochs. It can also ask to see (re-)encryptions of (the challenge) messages. If it asks to see the challenge ciphertext in some epoch i and corrupts a secret key in some epoch j such that either $j = i$ or it has obtained all re-keys $\mathtt{rk}_{i,i+1}, \ldots, \mathtt{rk}_{j-1,j}$ then it loses. Otherwise, the adversary wins if it guesses the bit b chosen by the challenger that determines whether $\mathtt{m}_0$ or $\mathtt{m}_1$ was encrypted.

Algorithm 4.16: Game KRot for key rotation using a (unidirectional multi-hop) PRE scheme PRE.

Challenger $\mathsf{KRot}^b(1^\lambda, 1^N)$
1 Set $\mathcal{C}, \mathcal{E}, \mathcal{S} = \emptyset$ `// Stores corrupted keys (`$\mathcal{C}$`), re-keys (`$\mathcal{E}$`) and revealed challenges (`$\mathcal{S}$`)`
2 Set $\mathcal{L}, \mathcal{L}^* = \emptyset$ `// Stores ciphertexts and challenge ciphertexts`
3 Set $ctr = 0$ `// Counter for generated ciphertexts`
4 $\mathtt{pp} \leftarrow \mathsf{PRE.S}(1^\lambda, 1^N)$ `//` N `determines the number of levels`
5 $(\mathtt{pk}_1, \mathtt{sk}_1), \ldots, (\mathtt{pk}_N, \mathtt{sk}_N) \leftarrow \mathsf{PRE.K}(\mathtt{pp})$
6 $\forall i \in [2, N] : \mathtt{rk}_{i-1,i} \leftarrow \mathsf{PRE.RK}((\mathtt{pk}_{i-1}, \mathtt{sk}_{i-1}), \mathtt{pk}_i)$
7 $b' \leftarrow \mathsf{A}^{(\mathtt{corrupt},\cdot),(\mathtt{rekey},\cdot),(\mathtt{encrypt},\cdot,\cdot),(\mathtt{challenge},\cdot,\cdot,\cdot),(\mathtt{reveal},\cdot,\cdot)}(\mathtt{pp}, \mathtt{pk}_1, \ldots, \mathtt{pk}_N)$
8 **if** $\exists i \in \mathcal{S}\, \exists j \in \mathcal{C} : i$ *is connected to* j **then**
9 **return** 0
10 **return** b'

Oracle $(\mathtt{corrupt}, i)$
11 Add i to $\mathcal{C}$
12 **return** $\mathtt{sk}_i$

Oracle $(\mathtt{rekey}, i)$
13 Add $(i-1, i)$ to $\mathcal{E}$
14 **return** $\mathtt{rk}_{i-1,i}$

Oracle $(\mathtt{encrypt}, i, \mathtt{m})$
15 Increment *ctr*, compute $(\mathtt{c}_i, i) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_i, (\mathtt{m}, i))$
16 **for** $j = i+1 \ldots N$ **do**
17 $(\mathtt{c}_j, j) \leftarrow \mathsf{PRE.RE}(\mathtt{rk}_{j-1,j}, \mathtt{pk}_{j-1}, \mathtt{pk}_j, (\mathtt{c}_{j-1}, j-1))$
18 Increment *ctr*, add $(ctr, i, \mathtt{m}, (\mathtt{c}_i, \ldots, \mathtt{c}_N))$ to $\mathcal{L}$

Oracle $(\mathtt{challenge}, i, (\mathtt{m}_0^*, \mathtt{m}_1^*))$ `// Single access`
19 Compute $(\mathtt{c}_i, i) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_i, (\mathtt{m}_b^*, i))$
20 **for** $j = i+1 \ldots N$ **do**
21 $(\mathtt{c}_j, j) \leftarrow \mathsf{PRE.RE}(\mathtt{rk}_{j-1,j}, \mathtt{pk}_{j-1}, \mathtt{pk}_j, (\mathtt{c}_{j-1}, j-1))$
22 Increment *ctr*, add $(ctr, i, \mathtt{m}_b^*, (\mathtt{c}_i, \ldots, \mathtt{c}_N))$ to $\mathcal{L}$ and *ctr* to $\mathcal{L}^*$

Oracle $(\mathtt{reveal}, k, j)$
23 Retrieve $(k, i, \mathtt{m}, (c_i, \ldots, c_N))$ from $\mathcal{L}$
24 **if** $k \in \mathcal{L}^*$ **then** add j to $\mathcal{S}$
25 **return** $\mathtt{c}_j$

Definition 34 (KRot security)**.** A PRE scheme is (t, ϵ)-adaptively secure against key rotation attacks if $\mathsf{KRot}^0 \approx_{(t,\epsilon)} \mathsf{KRot}^1$, where KRot^b is defined in Algorithm 4.16.

Note that, as with all other notions, restricting the adversary to a single `challenge` call is without loss of generality: a standard hybrid argument shows that this notion implies a

Algorithm 4.17: Challenge oracle in shKRot^b_ℓ for $\ell \in [1, N]$.

Oracle $(\mathtt{challenge}, i, (\mathtt{m}^*_0, \mathtt{m}^*_1))$
1 Compute $(\mathtt{c}_i, i) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_i, (\mathtt{m}^*_b, i))$
2 **for** $j = i + 1 \ldots \ell$ **do**
3 $\quad (\mathtt{c}_j, j) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_j, (\mathtt{m}^*_b, j))$
4 **for** $j = \ell + 1 \ldots N$ **do**
5 $\quad (\mathtt{c}_j, j) \leftarrow \mathsf{PRE.RE}(\mathtt{rk}_{j-1,j}, \mathtt{pk}_{j-1}, \mathtt{pk}_j, (\mathtt{c}_{j-1}, j-1))$
6 Increment *ctr*, add $(\mathit{ctr}, i, \mathtt{m}^*_b, (\mathtt{c}_i, \ldots, \mathtt{c}_N))$ to $\mathcal{L}$ and *ctr* to $\mathcal{L}^*$

more general multi-challenge variant of this definition. (In the jth hybrid, the first $j-1$ calls $(\mathtt{challenge}, i, (\mathtt{m}_0, \mathtt{m}_1))$ are answered like $(\mathtt{encrypt}, i, \mathtt{m}_1)$ and the remaining ones like $(\mathtt{encrypt}, i, \mathtt{m}_0)$.)

The adaptive notion of key-rotation security we defined is not immediately implied by HRA security: in the former the adversary is allowed to ask for $\mathtt{m}_b$ to be encrypted under $\mathtt{pk}_1$, to see its re-encryption under $\mathtt{pk}_2$ and to corrupt $\mathtt{pk}_1$. This is not allowed by the HRA game, as a challenge ciphertext is immediately revealed by the oracle and the corresponding key must therefore not be corrupted. We show however that if the PRE scheme is source-hiding then HRA security implies key-rotation (KRot) security.

Theorem 15 (KRot security)**.** *Let $N \in \mathbb{N}$. Then a PRE scheme that is (t_1, ϵ_1)-*PRE-HRA*-secure (under multiple challenges) and (t_2, ϵ_2)-source-hiding is (t, ϵ)-KRot-secure, where*

$$t := \min(t_1, t_2) - t_{\mathsf{KRot}}, \quad \textit{and} \quad \epsilon := \epsilon_1 + (N-1)\epsilon_2,$$

where t_{KRot} denotes the time complexity of simulating game KRot*.*

Proof. To prove the theorem we show that if the encryption scheme is source-hiding then the game KRot is indistinguishable from a game where, instead of being re-encrypted, the challenge is freshly encrypted when the adversary queries its `reveal` oracle on it.

Consider the hybrid game shKRot^b_ℓ defined similar to KRot^b, except that the challenge oracle is defined as in Algorithm 4.17. The difference between $\mathsf{shKRot}^b_{\ell-1}$ and shKRot^b_ℓ is whether the challenge oracle computes $\mathtt{c}_\ell$ by reencrypting or by freshly encrypting. An adversary distinguishing $\mathsf{shKRot}^b_{\ell-1}$ from shKRot^b_ℓ can thus be turned into an adversary against source-hiding (Definition 24): the reduction receives $(\mathtt{pp}, (\mathtt{pk}, \mathtt{sk}), (\mathtt{pk}', \mathtt{sk}'), \mathtt{rk})$, simulates the experiment setting $(\mathtt{pk}_{\ell-1}, \mathtt{sk}_{\ell-1}) := (\mathtt{pk}, \mathtt{sk})$, $(\mathtt{pk}_\ell, \mathtt{sk}_\ell) := (\mathtt{pk}', \mathtt{sk}')$, and $\mathtt{rk}_{\ell-1,\ell} := \mathtt{rk}$. When the adversary queries $(\mathtt{challenge}, i, (\mathtt{m}^*_0, \mathtt{m}^*_1))$, the reduction first computes for $j = i+1 \ldots \ell-2$: $(\mathtt{c}_j, j) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_j, (\mathtt{m}^*_b, j))$; it then queries $(\mathtt{challenge}, \mathtt{m}^*_b, \ell-1)$ to receive $\mathtt{c}_{\ell-1}$ (a fresh encryption) and $\mathtt{c}_\ell$ (a re-encryption of $\mathtt{c}_{\ell-1}$ or a fresh encryption depending on the challenger); for $j = \ell+1 \ldots N$ it computes $(\mathtt{c}_j, j) \leftarrow \mathsf{PRE.RE}(\mathtt{rk}_{j-1,j}, \mathtt{pk}_{j-1}, \mathtt{pk}_j, (\mathtt{c}_{j-1}, j-1))$. The reduction continues the simulation of the game (it has all secret keys) and returns whatever the adversary does.

shKRot^b_1 is the original game KRot^b, whereas shKRot^b_N is a game where the challenge message is freshly encrypted for every $\mathtt{pk}_i$. Moreover, shKRot^b_N is equivalently distributed to a game shKRot^b_{N+1} where the challenge oracle does not compute anything yet, and the reveal oracle directly computes challenge ciphertexts. shKRot^b_{N+1} is thus defined like KRot^b except that challenge and reveal oracle are defined as in Algorithm 4.18. Game shKRot^b_{N+1} can now

Algorithm 4.18: Challenge and reveal oracle in shKRot^b_{N+1}.

Oracle $(\texttt{challenge}, i, (\mathtt{m}^*_0, \mathtt{m}^*_1))$
1 Increment *ctr*, add $(\mathit{ctr}, i, \mathtt{m}^*_b, \emptyset)$ to $\mathcal{L}$ and *ctr* to $\mathcal{L}^*$

Oracle $(\texttt{reveal}, k, j)$
2 Retrieve $(k, i, \mathtt{m}, (\mathtt{c}_i, \ldots, \mathtt{c}_N))$ from $\mathcal{L}$ // $(\mathtt{c}_i, \ldots, \mathtt{c}_N)$ could be empty
3 **if** $k \in \mathcal{L}^*$ **then**
4 Add j to $\mathcal{S}$
5 Compute $(\mathtt{c}_j, j) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_j, (\mathtt{m}, j))$
6 **return** $\mathtt{c}_j$

be simulated for an adversary A in a straightforward way by a reduction R that plays (the multi-challenge version of) game HRA^b: R forwards all `corrupt` and `rekey` queries; when A makes a query `encrypt`, then R generates $(\mathtt{c}_i, \mathtt{c}_{i+1}, \ldots, \mathtt{c}_N)$ as follows: it queries its `encrypt` oracle to get $\mathtt{c}_i$ and its `reencrypt` oracle to obtain $\mathtt{c}_{i+1}, \ldots, \mathtt{c}_N$. (Note that this does not lead to any edges being added to $\mathcal{E}$.) When A makes its query `challenge` then R faithfully simulates shKRot^b_{N+1} and stores the message in $\mathcal{L}$ and the current counter value in $\mathcal{L}^*$. Queries $(\texttt{reveal}, k, j)$ with $k \notin \mathcal{L}^*$ are answered as specified in shKRot^b_{N+1}; if $k \in \mathcal{L}^*$ then R queries $(\texttt{challenge}, j, (\mathtt{m}^*_0, \mathtt{m}^*_1), j)$ and forwards the reply. Finally R returns A's final output b'.

R's probability of winning KRot^b is the same as A's probability of winning shKRot^b_{N+1}, as R only violates the winning condition that no key that was challenged is connected to a corrupt key via re-keys (represented by edges in $\mathcal{E}$ in both games) if and only if A does in game shKRot^b_{N+1}. □

4.7 Open Problems

We leave as open problems to find adaptively secure PRE schemes (either via the Piecewise-Guessing framework or using a new technique) for more general settings, which includes unidirectional PRE on general graphs, bidirectional PRE and CCA-secure PRE (the schemes above only satisfy CPA, and the slightly stronger HRA security notion).

Here we want to point forward to Chapter 7: The results there indicate that adaptive CPA security for unidirectional PRE on general DAGs cannot be derived from the properties of ciohertext indistinguishability and weak key-privacy alone, but other assumptions on the PRE scheme would be necessary.

CHAPTER 5

Adaptively Secure Continuous Group Key Agreement

5.1 Introduction

Messaging systems allow for asynchronous communication, where parties need not be online at the same time. Messages are buffered by an untrusted delivery server, and then relayed to the receiving party when it comes online. Secure messaging protocols (like Open Whisper Systems' Signal Protocol) provide end-to-end privacy and authenticity but, by having parties perform regular key updates, also stronger security guarantees like forward secrecy (FS) and post-compromise security (PCS). Here, FS means that even if a party gets compromised, previously delivered messages (usually all messages prior to the last key update) remain private. In turn, PCS guarantees that even if a party was compromised and its state leaks, normal protocol execution after the compromise ensures that eventually (usually after the next key update) future messages will again be private and authenticated.

Most existing protocols were originally designed for the two party case and do not scale beyond that. Thus, group messaging protocols are usually built on top of a complete network of two party channels. Unfortunately, this means that message sizes (at least for the crucial key update operations) grow linearly in the group size. In view of this, constructing messaging schemes that provide strong security – in particular FS and PCS – while *efficiently*[1] scaling to larger groups is an important but challenging open problem. Designing such a protocol is the ongoing focus of the IETF working group Message Layer Security (MLS) [mls].

Instead of constructing a messaging scheme directly, a modular approach seems more natural. For the two party case this was done by Alwen et al. [ACD19]. In this work we consider the concept of Continuous Group Key Agreement (CGKA), a generalization of their Continuous Key Agreement (CKA) to groups. Such a primitive can then be used to build a group messaging protocol as in [ACD19].

This Chapter essentially replicates, with permission, large parts of the full version [ACC+19] of our publication [KPW+21], © 2021 IEEE, https://doi.org/10.1109/SP40001.2021.00035.

[1] The meaning of efficient here will be determined by what can be implemented and receive adoption by the general public, but we think of it as having message sizes (poly)logarithmic in the size of the group.

5.1.1 Continuous Group Key Agreement

Informally, in a CGKA protocol any party ID_1 can initialise a group $\mathcal{U} = (\mathsf{ID}_1, \ldots, \mathsf{ID}_n)$ by sending a message to all group members, from which each group member can compute a shared group key $\mathtt{K}$. ID_1 must know a public key $\mathtt{pk}_i$ of each invitee ID_i, which in practice could be realized by having a key-server where parties can deposit their keys. As this key-management problem is largely orthogonal to the construction of a CGKA, we will assume that such an infrastructure exists.

Apart from initialising a group, CGKA allows any group member ID_i to *update* its key. Informally, after an Update[2] operation the state of ID_i is secure even if its previous state completely leaked to an adversary. Moreover, any group member can *add* a new party, or *remove* an existing one.

These operations (Update, Add, Remove) require sending a message to all members of the group. As we do not assume that the parties are online at the same time, ID_i cannot simply send a message to ID_j. Instead, all protocol messages are exchanged via an untrusted delivery server. Although the server can always prevent any communication taking place, we require that the shared group key in the CGKA protocol – and thus the messages encrypted in the messaging system built upon it – remains private.

Another issue we must take into account is the fact that (at least for the protocols discussed below) operations must be performed in the same order by all parties in order to maintain a consistent state. Even if the delivery server is honest, it can happen that two parties try to execute an operation at the same time. In this case, an ordering must be enforced, and it is natural to let the delivery server do it. Whenever a party wants to initiate an Update/Remove/Add operation, it sends the message to the delivery server and waits for an answer. If it gets a confirmation, it updates its state and deletes the old one. If it gets a reject, it deletes the new state and keeps the old one. Note that when a party gets corrupted while waiting for the confirmation, both the old and new state are leaked.

The formalization of CGKA is fairly recent, having first been introduced in [ACDT20]. In particular, the MLS working group predates it, which only complicates any account of its development. We try to give a brief overview below. Up to the writing of this paper, the MLS protocol has seen 9 versions, through which we can find two different CGKA protocols: ART and TreeKEM, both of which were incorporated as candidates in the initial MLS protocol draft. ART was later removed in the second version of the protocol, with TreeKEM (which has seen several modifications throughout the different versions) being the current candidate. Accordingly, we will refer to the CGKA construction underlying version X of the MLS draft as TreeKEMvX. We will also loosely use the term TreeKEM when referring to aspects that are not unique to specific versions or when there is no ambiguity.

Asynchronous Ratcheting Tree (ART). The first proposal of (a simplified variant of) a CGKA is the Asynchronous Ratcheting Tree (ART) by Cohn-Gordon et al. [CCG+18]. This protocol (as well as TreeKEM and the protocol formalized in this paper) identifies the group with a binary tree where edges are directed from the leaves to the root.[3] Each party ID_i in the group is assigned their own leaf, which is labelled with an ElGamal secret key x_i (known

[2] We use capital letters to refer to the operations (as opposed to verbs).

[3] The non standard direction of the edges here captures that knowledge of (the secret key of) the source node implies knowledge of the (secret key of the) sink node. Note that nodes therefore have one child and two parents.

only to ID_i) and a corresponding public value g^{x_i}. The values of internal nodes are defined recursively: an internal node whose two parents have secret values a and b has the secret value g^{ab} and public value $g^{\iota(g^{ab})}$, where ι is a map to the integers. The secret value of the root is the group key. As illustrated in Figure 5.1, a party can update its secret key x to a new key x' by computing a new path from x' to the (new) root, and then send the public values on this new path to everyone in the group so they can switch to the new tree. Note that the number of values that must be shared equals the depth of the tree, and thus (as the tree is balanced) is only logarithmic in the size of the group.

The authors prove the ART protocol secure even against adaptive adversaries. However, in this case, their reduction loses a factor that is super-exponential in the group size. To get meaningful security guarantees based on this reduction one requires a security parameter for the ElGamal scheme that is super-linear in the group size, resulting in large messages and defeating the whole purpose of using a tree structure.

TreeKEM. The TreeKEM proposal [BBR18, BBM+20] is similar to ART, as a group is still mapped to a balanced binary tree where each node is assigned a public and secret value. In TreeKEM those values are the public/secret key pair for an arbitrary public-key encryption scheme. As in ART, each leaf is assigned to a party, and only this party should know the secret key of its leaf, while the secret key of the root is the group key. Unlike in ART, TreeKEM does not require any relation between the secret key of a node and the secret key of its parent nodes. Instead, an edge $u \to v$ in the tree (recall that edges are directed and pointing from the leaves to the root) denotes that the secret key of v is encrypted under the public key of u. This ciphertext can now be distributed to the subset of the group who knows the secret key of u to convey the secret key of v to them. We will refer to this as "encrypting v to u". Below we will outline a slightly simplified construction, close to TreeKEMv7, which will later ease the understanding of the protocol here proposed.

To initialise a group, the initiating party creates a tree by assigning the leaves to the keys of the invited parties. She then samples fresh secret/public-key pairs for the internal nodes of the tree and computes the ciphertexts corresponding to all the edges in the tree. (Note that leaves have no ingoing edges and thus the group creator only needs to know their public keys.) Finally she sends all ciphertexts to the delivery server. If a party comes online, it receives the ciphertexts corresponding to the path from its leaf to the root from the server, and can then decrypt (as it has the secret key of the leaf) the nodes on this path all the way up to the group key in the root.

As illustrated in Figure 5.1, this construction naturally allows for adding and removing parties. If ID_i wants to remove ID_j, she simply samples a completely fresh path from a (fresh) leaf to a (fresh) root replacing the path from ID_j's leaf to the root. She then computes and shares all the ciphertexts required for the parties to switch to this new path *except* the ciphertext that encrypts to ID_j's leaf. ID_i can add ID_j similarly, she just samples a fresh path starting at a currently not occupied leaf, using ID_j's key as the new leaf node, and communicates the new keys to ID_j. This process can be optimized if the keys are derived hierarchically from a hash chain of seeds, so that a single seed needs to be encrypted to each party.

Unfortunately, adding and removing parties like this creates a new problem. After ID_i added or removed ID_j, it knows all the secret keys on the new path (except the leaf). To see why this is a problem, assume ID_i is corrupted while adding (or removing) ID_j (and no other corruptions ever take place), and later – once the adversary loses access to ID_i's state – ID_i executes an Update. Assume we use a naïve protocol where this Update replaces all the keys on the path

from ID_i's leaf to the root (as in ART) but nothing else. As ID_i's corruption also leaked keys not on this path, thus not replaced with the Update, the adversary will potentially still be able to compute the new group key, so the Update failed to achieve PCS.

To address this problem, TreeKEM introduced the concept of *blanking*. In a nutshell, TreeKEM wants to maintain the invariant that parties know only the secrets for nodes on the path from their leaf to the root. However, if a party adds (or removes) another party as outlined above, this invariant no longer holds. To fix this, TreeKEM declares any nodes violating the invariant as not having any secret (nor public) value assigned to them. Such nodes are said to be "blanked", and the protocol basically specifies to act as if the child of a blank node is connected directly to the blanked node's parents. In particular, when TreeKEM calls for encrypting something to a blank node, users will instead encrypt to this node's parents. In case one or both parents are blanked, one recurses and encrypts to their parents and so forth.

This saves the invariant, but hurts efficiency, as we now no longer consider a binary tree and, depending on the sequence of Adds and Removes, can end up with a "blanked" tree that has effective indegree linear in the number of parties. The reason one can still hope for TreeKEM's efficiency to not degrade too much and stay close to logarithmic in practice comes from the fact that blanked nodes can heal: whenever a party performs an Update operation, all the blank nodes on the path from its leaf to the root become normal again.

For completeness, we mention that the design of TreeKEMv9 differs in essentially two aspects. First, operations are not executed standalone, but bundled into groups: users can at any point *propose* an operation, not having any impact on the group state; then, a user ID_j can collect those proposals and execute them at once in a *Commit*, which includes an update of ID_j's path, and moves the group forward into a new epoch. This allows e.g. for ID_i to propose an Update by just sending their new leaf public key and waiting for someone else to commit that proposal (which will in turn blank ID_i's path). Second, Adds no longer involve blanking: a new user's leaf node will be directly connected to the root, and progressively pushed down the tree as users within the appropriate subtree commit. In particular, the initialization of the tree will now consist of a Commit including Add proposals for each of the group members. Since none of these aspects help in the understanding of the proposed protocol, we omit the details and refer the reader to the MLS draft [BBM+20].

5.1.2 Our Contribution

We formalize an alternative CGKA design, stemming from TreeKEM, first proposed by Millican on the MLS mailing list in February 2018[4], which we call Tainted TreeKEM, or TTKEM for short. In our publication [KPW+21] we compare the efficiency of TTKEM and TreeKEM by running simulations and show TTKEM to be more efficient than TreeKEM in certain realistic scenarios; this is not part of this thesis and we refer to [KPW+21] for details. Furthermore, we prove TTKEM to satisfy a comprehensive security statement which captures the intuition that an Update fixes a compromised state. Our proof can easily be adapted to TreeKEM, for which one can get a similar security statement, and constitutes the first adaptive security proof with subexponential loss for any CGKA protocol. In the following we elaborate on these contributions.

[4] [MLS] Removing members from groups, Jon Millican {jmillican@fb.com}, 12 February 2018, `https://mailarchive.ietf.org/arch/msg/mls/4-gvXpc-LGbWoUS7DKGYG65lkxs`

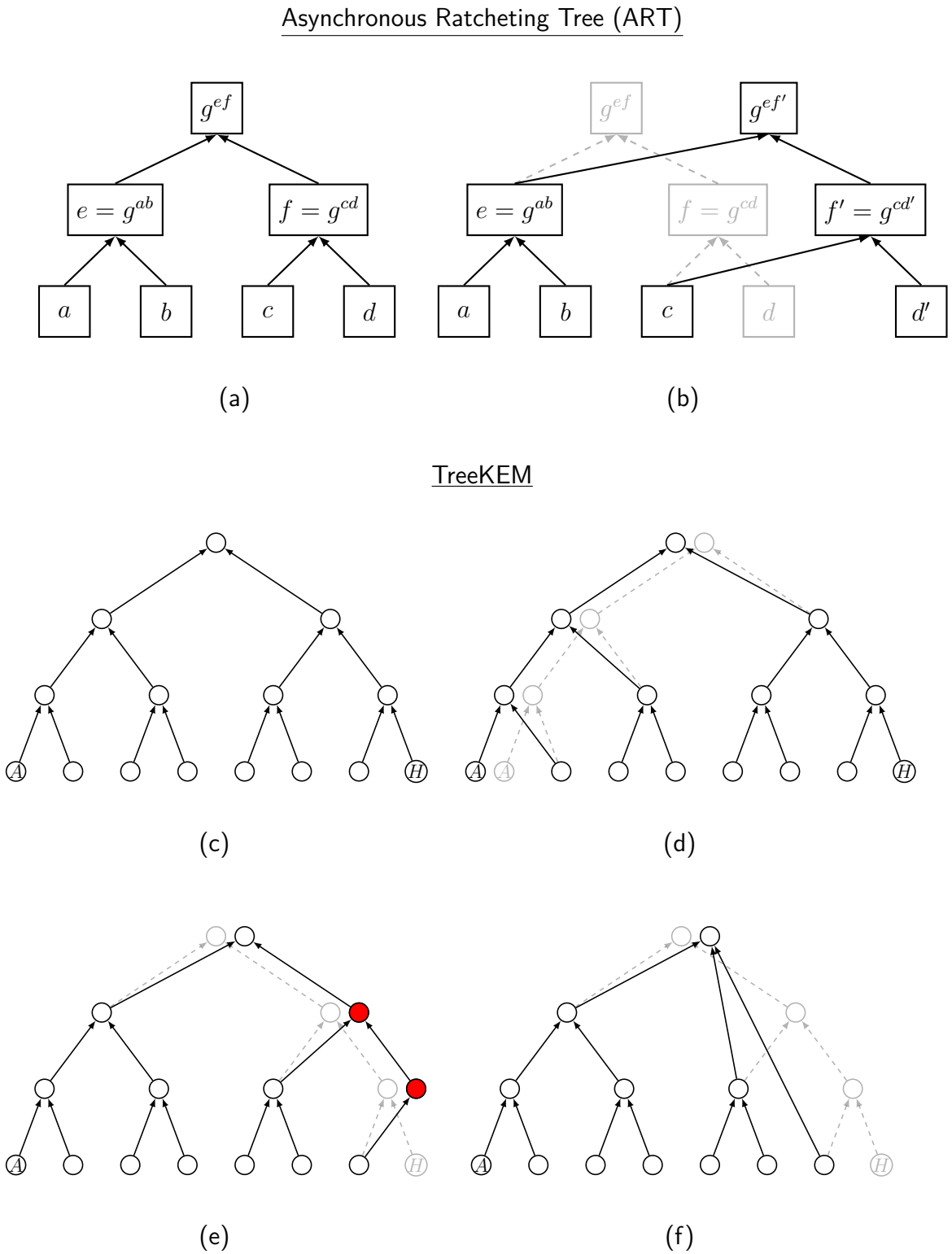


Figure 5.1: **Top**: Illustration of an Update in the ART protocol. The state of the tree changes from (a) to (b) when Dave (node d) updates his internal state to d'. **Bottom**: Update and Remove in naïve TreeKEM and TreeKEM with blanking. The state of a completely filled tree is shown in (c). The state changes from (c) to (d) when Alice (node A) performs an Update operation. This changes to (e) when Alice removes Harry (node H) in naïve TreeKEM (with the nodes that Alice should not know in red) or to (f) in the actual TreeKEM protocol which uses blanking.

Tainted TreeKEM (TTKEM). As just outlined, the reason TreeKEM can be inefficient comes from the fact that once a node is blanked, we cannot simply encrypt to it, but instead must encrypt to both its parents, if those are blanked, to their parents, and so forth. The rationale for blanking is to enforce an invariant which states that the secret key of any (non-blanked) node is only known to parties whose leaves are ancestors of this node. This seems overly paranoid, assume Alice removed Henry as illustrated in Figure 5.1, then the red nodes must be blanked as Alice knows their value, but it is instructive to analyze when this knowledge becomes an issue if no blanking takes place: If Alice is not corrupted when sending the Remove operation to the delivery server there is no issue as she will delete secret keys she should not know right after sending the message. If Alice is corrupted then the adversary learns those secret keys. But even though now the invariant doesn't hold, it is not a security issue as an adversary who corrupted Alice will know the group key anyway. Only once Alice updates (by replacing the values on the path from her leaf to the root) there is a problem, as without blanking not all secret keys known by the adversary are replaced, and thus the adversary will be able to decrypt the new group key; something an Update should have prevented (more generally, we want the group key to be safe once all the parties whose state leaked have updated).

Keeping dirty nodes around, tainting versus blanking. In TTKEM we use an alternative approach, where we do not blank nodes, but instead keep track of which secret keys of nodes have been created by parties who are not supposed to know them. Specifically, we refer to nodes whose secret keys were created by a party ID_i which is not an ancestor of the node as *tainted (by* ID_i*)*. The group keeps track of which nodes are tainted and by whom. A node tainted by ID_i will be treated like a regular node, except for the cases where ID_i performs an Update or is removed, in which it must get updated. Let us remark that tainted nodes can heal similarly to blanked nodes: once a party performs an Update, all nodes on the path from its leaf to the root are no longer tainted.

Efficiency of TTKEM vs TreeKEM. Efficiency-wise TreeKEM and TTKEM are incomparable. Depending on the sequence of operations performed either TreeKEM or TTKEM can be more efficient (or they can be identical). Thus, which one will be more efficient in practice will depend on the distribution of operation patterns we observe. In [KPW+21] we show that for some natural cases TTKEM will significantly outperform TreeKEM. This improvement is most patent in the case where a small subset of parties perform most of the Add and Remove operations. In practice, this could correspond to a setting where we have a small group of administrators who are the only parties allowed to add/remove parties. The efficiency gap grows further if the administrators have a lower risk of compromise than other group members and thus can be required to update less frequently. In this setting, TTKEM approaches the efficiency of naïve TreeKEM.

When we compare the efficiency of the CGKA protocols we focus on the number of ciphertexts a party must exchange with the delivery server for an (Update, Add or Remove) operation. The reason for this is that the alternative metric of measuring the number of ciphertexts a party needs to download to process an operation is not as relevant, since, for all protocols considered, this number will be logarithmic in the worst case.[5] We refer to [KPW+21] for the details and precise results of our efficiency analysis.

[5]There is, however, still room for improvement in the case where a group member comes online and must process a large number of operations, as these could potentially be somehow batched by the server.

Security of (Tainted) TreeKEM. A main contribution of our publication [KPW+21] is a security proof for TTKEM for a *comprehensible* security statement that intuitively captures how Updates ensure FS and PCS, in a *strong* security model. In particular, this constitutes the first adaptive security proof for any TreeKEM-related protocol. Moreover, both the security statement and the proof can be easily adapted to TreeKEM. We elaborate in the following section.

5.1.3 The Adversarial Model

We anticipate an adversary who works in rounds, in each round it may adaptively choose an action, including start/stop corrupting a party, instruct a party to initalize an operation or relay a message. The adversary can choose to corrupt any party, after which its state becomes fully visible to the adversary. In particular, corrupting a party gives the adversary access to the random coins used by said party when executing any group operation, deeming the party's actions deterministic in the eyes of the adversary throughout the corruption. We would like to stress that security in this strong model implies security in weaker and potentially more realistic models, e.g. consider the setting where malware in a device leaks some of the randomness bits but cannot modify them. He can also choose to stop the corruption of a currently corrupted party. The adversary can instruct a party to initalize an Init/Update/Remove/Add operation. This party then immediately outputs the corresponding message to be sent to the delivery server. The goal of the adversary is to break the security of a group key (i.e., a secret key that is contained in the root in the view of at least one party) that – given the sequence of actions performed – it should not trivially know.

We now discuss different possible restrictions on the adversary corresponding to qualitatively different levels of security.

Adaptiveness. The literature distinguishes between *selective* and *adaptive* adversaries. In the selective case, an adversary is required to make all or some of its choices (here this means the sequence of operations and which key it is going to break) at the beginning of the security experiment, without seeing any public keys or the results of previous actions. While it is often more convenient to prove security in this setting, it is clearly unrealistic, since in the real world adversaries may adjust their behaviour based on what they observe during the attack. So obviously, security against adaptive adversaries is desirable. There is a generic reduction from selective to adaptive adversaries that simply guesses what the adversary may choose (this is the approach effectively taken in [CCG+18]). However, this involves a loss in the advantage that is exponential (or even superexponential) in the size of the group. This means that in order to *provably* achieve meaningful security, one needs to set the underlying security parameter linear in the group size, which results in the Update messages having size linear in the group size (since they usually consist of encryptions of secret keys). But the trivial construction based on pairwise channels also has message size that is linear in the number of group members, so such a security proof defeats the whole purpose of the protocol: having small Update messages! The adversaries we consider are adaptive while the security loss we achieve is only quasipolynomial (or even polynomial) in the standard model (in the random oracle model, resp.; see details below).

Activeness. One can classify adversaries with respect to their power to interact with the protocol during the attack. For example, the weakest form of adversary would be a *passive* adversary, i.e. an eavesdropper that only observes the communication but does not alter

	Constraints on PKE	Forward Secrecy	Adversary	Tightness	
				Selective	Adaptive
ART [CCG+18]	ElGamal (any with contributive NIKE)	Standard	Passive	$O(M)$ RO	$\Omega\left((MQ)^M\right)$ RO
TreeKEM [BBM+20]**	None	Standard	Passive	$\mathcal{O}(M)$ S	$\Omega\left((MQ)^M\right)$ S*
rTreeKEM [ACDT20]	UPKE	Strong	Passive	$\mathcal{O}(M)$ RO	$\Omega\left((MQ)^M\right)$ RO*
TTKEM (this work)	None	Standard	Part. active	$\mathcal{O}(M)$ S,RO	$\mathcal{O}\left(M^2Q^{\log(M)}\right)$ S $\mathcal{O}\left((MQ)^2\right)$ RO

Table 5.1: Table depicting the different security levels satisfied by CGKA protocols. M, Q denote upper bounds on the number of group members and Init/Update/Remove/Add queries, respectively. The first two columns correspond to protocol characteristics, and the right-most three to the best known proofs. S and RO stand Standard and Random Oracle Models, respectively. (*) These would follow from the selective proof via a straightforward complexity leveraging argument. Such an argument is implicit in the proof of [CCG+18]. (**) [ACDT20] provides a sketch for a proof against passive adaptive adversaries with a quasi-polynomial loss in the SM; this paper suggests proofs against part. active, adaptive adversaries both in the SM and ROM, with the same tightness as the ones for TTKEM.

any messages, whereas the strongest notion would be an *active* adversary who can behave completely arbitrarily. In this work we consider "partially" active adversaries who can arbitrarily schedule the messages of the delivery server, and thus force different users into inconsistent states. But we do not consider adversaries who can arbitrarily deviate and for example use secret keys of corrupted parties to create malformed messages. Restricting to partially active adversaries is fairly common in this setting [ACD19, CHK19, JMM19] (also somewhat implied by the model of [DV19], where communication must halt after an active attack). Achieving or even defining meaningful security against fully active adversaries is the subject of ongoing research [ACJM20].

Forward Secrecy. FS (and PCS) are standard notions expected to hold in modern messaging protocols. However, in contrast to the two-party setting, formalizing FS in the group setting is more nuanced. One natural notion is to require that a key is secure if all parties have performed an update before being corrupted. This is the notion considered in [CCG+18] and the one we adopt here and call it *standard FS*. In contrast, [ACDT20] defines a stronger notion we refer to as *strong FS*. It requires keys to be secure as soon as possible subject to not violating basic completeness of the CGKA protocol. However, this is only required in executions where protocol packets are delivered in the same order to all group members.[6] The construction in [ACDT20] in fact achieves strong FS, but only for adversaries that are much less active than ours. We provide some details in the next section.

[6]Going even further, the (efficient but impractical) CGKA protocols of [ACJM20] enjoys *optimal FS*. That is, keys must become secure as soon as possible for *arbitrary* delivery order. In fact, two of their protocols enjoy optimal FS even against adversaries that can arbitrarily manipulate and generate traffic; a type of active security even stronger than the one considered in this work.

The safe predicate. Providing PCS and FS requires to clearly define which keys we expect to be secure given a sequence of adversarial actions. Given the asynchronous setting where group members might be in different states, and an active adversary that may force users into inconsistent states, this is quite involved. Note that group members might even have different views of who is currently a member of the group. We give a compact and intuitive predicate that captures exactly what PCS and FS guarantees TTKEM provides.

The reduction in the standard model via piecewise guessing. Recall that there is a trivial reduction between selective and adaptive adversaries that simply guesses the necessary information and fails if the guess was incorrect. This loses an exponential factor in the amount of information that needs to be guessed. In Chapter 3 we proposed a general framework (refered to as *Piecewise-Guessing*) that allows to reduce this loss under certain conditions. The resulting loss depends on the graph structure that naturally arises from the security experiment. Applying the framework in the obvious way (which already requires non-trivial effort) we achieve a quasipolynomial security loss $\approx (M \cdot Q)^{2\log(M)}$ against partially active adversaries, where M is an upper bound on the number of group members and Q is the number of Init/Update/Remove/Add queries the adversary issues. Using a more careful analysis and taking the more restrictive structure of the queries and the graph constructed in the CGKA security game for TTKEM into account, we can improve this to $\approx Q^{\log(M)}$. We note that all steps of the proof strategy also apply to TreeKEM, and so an equivalent proof for it would easily follow.

The reduction in the random oracle model. In (Tainted) TreeKEM, a node is identified with a short seed s, from which the public/secret key pairs of this node are derived. If the randomness used to sample those keys is a hash of s, and we model this hash as a random oracle, we can give a much better *polynomial* bound for the adaptive security of TTKEM.

This proof is very different from the proof in the standard model and does not use the Piecewise-Guessing framework. Some of the techniques resemble a security proof of Logical Key Hierarchies (cf. Section 5.1.4) by Panjwani [Pan07], but otherwise the proof is entirely self-contained and novel. Again, our proof can also be applied to TreeKEM. As a sidenote, we prove and employ a new result on a *public-key* version of GSD in the random oracle model, which we believe to be of independent interest.

5.1.4 Related Work

The basic idea of TreeKEM can be traced back to Logical Key Hierarchies (LKH) [WHA98, WGL98, CGI+99]. These were introduced as an efficient solution to multicast key distribution (MKD), where a trusted and central authority wants to encrypt messages to a dynamically changing group of receivers. Clearly, the main difference to continuous group key agreements is the presence of a central authority that distributes the keys to users and may add and remove users. At the heart of TreeKEM is the realization that if one replaces symmetric key encryption with public key encryption in LKH, then any group member can perform the actions that the central authority does in MKD. But, as described above, this introduces the problem that some users now know the secret keys in parts of the tree they are not supposed to, which creates security problems. This is where the main novelties of TreeKEM and follow up work lies: in providing mechnanisms to achieve PCS and FS nonetheless.

LKH has been proven secure even against adaptive adversaries with a quasi-polynomial bound [Pan07]. Unfortunately, there are several important differences between LKH that do not allow

us to simply rely on [Pan07] to prove TTKEM or TreeKEM secure: 1) their proof is in the symmetric key setting, while we are using public key encryption; 2) their proof assumes a central authority and there is no concept of PCS or FS; 3) for efficiency reasons, TTKEM and TreeKEM use hierarchical key derivation, which the proof in [Pan07] does not take into account (even though it had already been proposed in optimized versions of LKH [CGI+99]) and it is a priori unclear how this affects the proof; 4) we are also interested in proving security in the random oracle model, which, as we show, gives tighter bounds.

Since the appearance of the double ratchet algorithm [PM16], implemented in applications like Signal or Whatsapp, secure messaging has received a lot of attention, particularly in the two party case [BSJ+17, DV19, JS18, JMM19, PR18, ACD19]. In the group setting, the main example of such a protocol is TreeKEM [BBR18, BBM+20], currently in development by the IETF MLS working group. Its predecesor was the ART protocol [CCG+18], whose proposal motivated the creation of the mentioned working group. A study of PCS in settings with multiple groups was done by Cremers et al. [CHK19], and Weidner [Mat19] explored a variant of TreeKEM allowing for less reliance on the server for correctness. In a follow-up work, Alwen et al. [ACJM20] study the security of CGKA protocols against insider attacks. Finally, in [AAB+21] we consider the setting of several overlapping groups and suggest a protocol for multicast encryption or CGKA for static subgroup structures.

rTreeKEM. Recently, Alwen et al. [ACDT20] introduced another variant of TreeKEM, termed re-randomized TreeKEM (rTreeKEM). Since their paper structure shares similarities with ours, we will discuss the differences between them.

First, it should be noted that the aims of the protocols are very different: while TTKEM seeks to improve the efficiency of TreeKEM by removing the need for blanks, rTreeKEM's focus is on improving its forward secrecy guarantees to achieve strong FS. However, we see no reason why one could not combine both protocols, endowing TTKEM with strong FS. Moreover, it seems plausible that the proof techniques developed in this work can also be applied to the rTreeKEM construction or to the combination of the two.

Second, their work already defines CGKA as an abstraction of the main problem TreeKEM aims to solve. We use their completeness notion, but add a *Confirm and Deliver* algorithm to their definition. The reason for this is that we work in the more general model that allows a malicious delivery server, i.e. the adversary can reorder and withhold messages at will. The model in [ACDT20] requires the delivery server to be basically honest: the server can delay, but never send inconsistent messages to parties, i.e. the adversary in [ACDT20] is almost passive.

Last, both works provide security proofs, albeit these differ considerably. Their paper provides proofs for both TreeKEM and rTreeKEM with a polynomial security loss, although these concern selective security only. They also sketch a security proof against adaptive adversaries losing a quasi-polynomial factor (for TreeKEM in the standard model, for rTreeKEM in the random oracle model). In contrast, we give formal proofs for the *adaptive* security of TTKEM with only polynomial loss in the random oracle model and quasi-polynomial in the standard model; and, as mentioned, against a stronger partially active adversary. Also, proofs with the same bounds would follow for TreeKEM.

5.1.5 Impact on MLS

As of writing, the current version of the MLS draft (MLS v9) differs substantially from TTKEM, mainly due to the Proposal-Commit structure. However, it should be noted that TTKEM can be cast in that same fashion, as it is indeed done in [ACJM20]. As with TreeKEM, the application of this framework would bring an efficiency tradeoff that should be studied carefully and which we leave for further work, though noting the challenge in doing so without real world data. As for our security proofs, a security proof for TreeKEM follows from the one given in the paper, so we believe this work to be of relevance to the MLS community.

5.2 Description of TTKEM

5.2.1 Asynchronous Continuous Group Key Agreement Syntax

Definition 35 (Asynchronous continuous group key agreement)**.** An *asynchronous continuous group key agreement* (CGKA) scheme is an 8-tuple of algorithms CGKA $=$ (keygen, init, add, rem, upd, dlv, proc, key) with the following syntax and semantics:

Key Generation**:** Fresh InitKey pairs are generated using $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{keygen}(1^\lambda)$ by users prior to joining a group. Public keys are used to invite parties to join a group.

Initialize a Group**:** For $i \in [2, n]$ let $\mathtt{pk}_i$ be an InitKey public key belonging to party ID_i. Let $\mathcal{U} = (\mathsf{ID}_1, \ldots, \mathsf{ID}_n)$. Party ID_1 creates a new group with membership $\mathcal{U}$ by running:

$$(\gamma, [\mathtt{W}_2, \ldots, \mathtt{W}_n]) \leftarrow \mathsf{init}\,(\mathcal{U}, [\mathtt{pk}_1, \ldots, \mathtt{pk}_n])$$

and sending *welcome message* $\mathtt{W}_i$ for party ID_i to the server. Finally, ID_1 stores its *local state* γ for later use.

Adding a Member**:** A group member with local state γ can add party ID to the group by running $(\gamma', \mathtt{W}, \mathtt{T}) \leftarrow \mathsf{add}(\gamma, \mathsf{ID}, \mathtt{pk})$ and sending *welcome message* $\mathtt{W}$ for party ID and the *add message* $\mathtt{T}$ for all group members (including ID) to the server. They store the old state γ and new *pending* state γ' until getting a confirmation from the delivery server as defined below.

Removing a Member**:** A group member with local state γ can remove group member ID by running $(\gamma', \mathtt{T}) \leftarrow \mathsf{rem}(\gamma, \mathsf{ID})$ and sending the *remove message* $\mathtt{T}$ for all group members (incl. ID) to the server and storing γ, γ'.

Update**:** A group member with local state γ can perform an update by running $(\gamma', \mathtt{T}) \leftarrow \mathsf{upd}(\gamma)$ and sending the *update message* $\mathtt{T}$ for all group members to the server and storing γ, γ'.

Confirm and Deliver**:** The delivery server upon receiving a (set of) CGKA protocol message(s) $\mathtt{T}$ (including welcome messages) generated by a party ID by running $\mathsf{dlv}(\mathsf{ID}, \mathtt{T})$ either sends $\mathtt{T}$ to the corresponding member(s) and sends a message confirm to ID, in which case ID deletes their old state γ and replaces it with the new pending state γ', or sends a message reject to ID, in which case ID deletes γ'.

Process**:** Upon receiving an incoming (set of) CGKA protocol message(s) $\mathtt{T}$ (including welcome messages) a party immediately processes them by running $(\gamma, \mathtt{K}) \leftarrow \mathsf{proc}(\gamma, \mathtt{T})$.

Get Group Key: At any point a party can extract the current group key $\mathtt{K}$ from their local state γ by running $(\gamma, \mathtt{K}) \leftarrow \mathsf{key}(\gamma)$.

We remark that while the protocol allows any group member to add a new party to the group as well as remove any member from the group it is up to the higher level message protocol (or even higher level application) to decide if such an operation is indeed permitted. (If not, then clients can always simply choose to ignore the add/remove message.) At the CGKA level, though, all such operations are possible.

5.2.2 Overview

In this work, a directed binary tree $B \in \mathcal{B}$ is defined recursively as a graph that is either the empty graph, a root node, or a root node whose parents are root nodes of trees themselves. Note that this corresponds to a standard definition of trees with reversed edges. We choose this definition since it is much more intuitive in our context and highlights the connection between the protocol and the GSD game used for the security proof (cf. Definition 41). Note that paths in the tree now start at leaves and end at the root node.

The nodes in the tree are associated with the following values:

- a seed Δ
- (all nodes except the root) a secret/public key pair derived deterministically from the seed: $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{Gen}(\Delta)$
- (only leaf nodes) a credential
- (all except leaves and root) a tainter ID

The root has no associated public/secret key pair, instead its seed is the current group key.

To achieve FS and PCS, and to manage group membership, it is necessary to constantly renew the secret keys used in the protocol. We will do this through the group operations **Update**, **Remove** and **Add**. We will use the term *refresh* to refer to the renewal of a particular (set of) key(s) (as opposed to the group operation). Each group operation will refresh a part of the tree, always including the root and thus resulting in a new group key which can be decrypted by all members of the current group. Users will also have a list of Initialization Keys (init keys) stored in some key-server, widely available and regularly updated, and used to add users to new groups.

Each group member should have a consistent view of the public information in the tree, namely public keys, credentials, tainter IDs and past operations. We assume that a party will only process operations issued by parties that (at the time of issuing) shared the same view of the tree. This can easily be enforced by adding a (collision-resistant) hash of the operations processed so far [DV19, JMM19][7]. Furthermore, group members will have a partial view of the secret keys. More precisely, every user has an associated *protocol state* $\gamma(\mathsf{ID})$ (or state for short when there is no ambiguity), which represents everything users need to know to stay part of the group (we implicitly assume a particular group, considering different groups' secrets independent). In particular, we define a state as the triple $\gamma(\mathsf{ID}) = (\mathcal{U}, B, h)$, where

- $\mathcal{U}$ denotes the set of group members, i.e. ID's that are part of the group
- B denotes a binary tree defined as above, where for each group member, their credential is associated to a leaf node.

[7] For efficiency reasons one could use a Merkle-Damgård hash so that from the hash of a (potentially long) string T we can efficiently compute the hash of T concatenated with a new operation t.

- h denotes the hash of the group transcript so far, to ensure consistency.

Each user also has a, typically empty, *pending state* $\gamma'(\mathsf{ID})$ which stores the updated group state resulting from the last issued group operation while they wait for confirmation.

As mentioned, a user will generally not have knowledge of the secret keys associated to all tree nodes. However, if they add or remove parties, they will potentially gain knowledge of secret keys outside their path. We observe that this will not be a problem as long as we have a mechanism to keep track of those nodes and refresh them when necessary – towards this end we introduce the concept of tainting.

Tainting. Whenever party ID_i refreshes a node not lying on their path to the root, that node becomes *tainted* by ID_i. Whenever a node is tainted by a party ID_i, that party has potentially had knowledge of its current secret in the past. So, if ID_i was corrupted in the past, the secrecy of that value is considered compromised (even if ID_i deleted that value right away and is no longer compromised). Even worse, all values that were encrypted to that node are compromised too. We will assign a tainter ID to all nodes. This can be empty, i.e. the node is untainted, or corresponds to a single party's ID, that who last generated this node's secret but is not supposed to know it. The tainter ID of a node is determined by the following simple rules.

- After ID initialises, all internal nodes not on ID's path become tainted by ID.
- If ID updates or adds/removes someone, refreshed nodes on ID's path become untainted.
- If ID updates or adds/removes someone, all refreshed nodes *not* on ID's path become tainted by ID.

Hierarchical derivation of updates. When refreshing a whole path we sample a seed Δ_0 and derive all the secrets for that path from it. This way, we reduce the number of decryptions needed to process the update, as parties only need to recover the seed for the "lowest" node that concerns them, and then can derive the rest locally. To derive the different new secrets we follow the specification of TreeKEMv9 [BBM+20]. Essentially, we consider a hash function H, fix two tags x_1 and x_2 and consider the two hash functions H_1, H_2 with $H_i(\cdot) = H(\cdot, x_i)$. Together with a Gen function that outputs a public/secret key pair, we derive the keys for the nodes as follows:

$$\begin{aligned} \Delta_{i+1} &:= H_1(\Delta_i) \\ (\mathtt{pk}_i, \mathtt{sk}_i) &\leftarrow \mathsf{Gen}(H_2(\Delta_i)), \end{aligned} \tag{5.1}$$

where Δ_i is the seed for the ith node (the leaf being the 0th node, its child the 1st etc.) on the path and $(\mathtt{pk}_i, \mathtt{sk}_i)$ its new key pair. For the proof in the standard model we only require H_i to be pseudorandom functions, with Δ_i the key and x_i the input.

With the introduction of tainting, it is no longer the case that all nodes to be refreshed lie on a path. Hence, we partition the set of all the nodes to be refreshed into paths and use a different seed for each path. For this we need a unique path cover, as users processing the update will need to know which nodes' secrets depend on which; see Figure 5.2. Any unambiguous partition suffices, the only condition required is that the updating of paths is done in a particular common order that allows for encryptions to to-be-refreshed nodes to be done under the respective updated public key (one cannot hope for PCS otherwise).

Let us stress that a party processing an update involving tainted nodes might need to retrieve and decrypt more than one encrypted seed, as the refreshed nodes on its path might not all

be derived hierarchically. Nonetheless, any party needs to decrypt at most $\log n$ ciphertexts in the worst case.

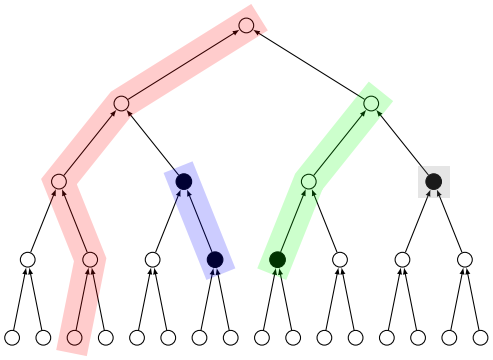


Figure 5.2: Path partition resulting from an update by Charlie (3^{rd} leaf node), with nodes tainted by him shown in black. To process it the grey node must be updated before the green path and the blue path before Charlie's (in red).

5.2.3 TTKEM Dynamics

Whenever a user ID_i wants to perform a group operation, they will generate the appropriate Initialize, Update, Add or Remove message, store the updated state resulting from processing such a message in γ', and send the appropriate information to the delivery server, which will then respond with a confirm or reject, prompting ID_i to move to state γ' (i.e. set $\gamma \leftarrow \gamma'$) or to delete γ', respectively. If the (honest) delivery server confirms an operation, it will also deliver it to all the group members, who will process it and update their states accordingly. Messages should contain the identity of the sender, the operation type, encryptions of the new seeds, any new public keys, and a hash h of the transcript so far, ensuring consistency. A more detailed description, as well as pseudo-code for the distinct operations is out of the scope of this work and can be found in the full version of our paper [ACC+19].

Initialize. To create a new group with parties $\mathcal{U} = \{\mathsf{ID}_1, \ldots, \mathsf{ID}_n\}$, a user ID_1 generates a new tree B, where the leaves have associated the init keys corresponding to the group members. The group creator then samples new key pairs for all the other nodes in B (optimizing with hierarchical derivation) and crafts welcome messages for each party. These welcome messages should include an encryption of the seed that allows the computation of the keys of the appropriate path, together with $\mathcal{U}$ and the public part of B.

Add. To add a new member ID_j to the group, ID_i identifies a free spot for them, hashes her secret key together with some freshly sampled randomness to obtain a seed Δ[8], and derives seeds for the nodes along the path to the root. She then encrypts the new seeds to all the nodes in the co-path (one ciphertext per node suffices given the hierarchical derivation) and sends them over together with the identity ID_j of the added party and the current transcript hash h. ID_i will also craft a welcome message for the added party containing an encryption of the appropriate seed, $\mathcal{U}$, h and the public part of B.

[8]This way the new keys will be secure against an adversary that does not have either knowledge of ID_i's secret key or control/knowledge of the randomness used.

Update. To perform an Update, a user computes a path partition for the set of nodes not on her path that need to be refreshed (nodes tainted or with a tainted ancestor), samples a seed per such path, plus a seed for their path, and derives the new key-pairs for each node, as described above. She then encrypts the secret keys under the appropriate public keys in the copaths and sends this information together with the current transcript hash h to the server.

Remove. To remove ID_j, user ID_i performs an Update as if it was ID_j, refreshing all nodes in ID_j's path to the root, as well as all her tainted nodes (which will become tainted by ID_i after the removal). Note that a user cannot remove herself. Instead, we imagine a user could request for someone to remove her and delete her state.

Process. When a user receives a protocol message `T`, she first checks whether the included transcript hash matches her own stored transcript hash h; if this is not the case, she ignores the message. Otherwise, she identifies which kind of message it is and performs the appropriate update of their state, by updating the list of participants if necessary, overwriting any keys, and updating the tainter ID's. If it is a *confirm* or a *reject*, i.e. it was an operation issued by herself, she updates the current state γ to γ' or simply deletes γ', respectively. Finally, she updates the transcript hash.

5.3 Security of TTKEM

We will prove security for TTKEM against fully adaptive, partially active adversaries, even when group members are in inconsistent states. In Section 5.3.1 we present the security game we consider and in Section 5.3.2 we present a simple predicate which allows to determine for which group keys we can guarantee security. The latter predicate incorporates the intuition that Updates allow a party to heal her state. It should be noted that we consider initialization keys as representing identities, as otherwise we would neglect some other cases which we would intuitively also consider secure, such as removing a corrupted party and adding them again once uncorrupted (this is secure per our predicate as they would be treated as a new identity, generated at the time the init key was).

Throughout our proofs, we only consider a single challenge per game for simplicity; a standard hybrid argument allows us to extend security to multiple challenges, with a loss linear in the number of challenges. In order to simulate extra challenges, an extra oracle that reveals group keys would be needed, but this would have no effect on the security proof - in particular GSD-like proofs already allow for the corruption of individual keys.

5.3.1 Security Model

Definition 36 (Asynchronous CGKA security)**.** The security for CGKA is modelled using a game between a challenger C and an adversary A. At the beginning of the game, the adversary A initialises the group $\mathcal{U}_0$ by querying `initialise`$(\mathsf{ID}_i, \mathcal{U}_0)$. The adversary A can then make a sequence of queries, enumerated below, in any arbitrary order. On a high level, `add` and `remove` allow the adversary to control the structure of the group, whereas the queries `confirm` and `process` allow it to control the scheduling of the messages. The query `update` simulates the refreshing of a local state. Finally, `start-corrupt` and `end-corrupt` enable the adversary to corrupt the users for a time period. The entire state (old and pending) and random coins of a corrupted user are leaked to the adversary during this period.

1. $\texttt{add}(\mathsf{ID}, \mathsf{ID}')$: a user ID requests to add another user ID' to the group.
2. $\texttt{remove}(\mathsf{ID}, \mathsf{ID}')$: a user ID requests to remove another user ID' from the group.
3. $\texttt{update}(\mathsf{ID})$: the user ID requests to refresh its current local state γ.
4. $\texttt{confirm}(q, \beta)$: the q-th query in the game, which must be an action $\mathsf{a} \in \{\texttt{add}, \texttt{remove}, \texttt{update}\}$ by some user ID, is either confirmed (if $\beta = 1$) or rejected (if $\beta = 0$). In case the action is confirmed, C updates ID's state and deletes the previous state; otherwise ID keeps its previous state).
5. $\texttt{process}(q, \mathsf{ID}')$: if the q-th query is as above, this action forwards the ($\texttt{W}$ or $\texttt{T}$) message to party ID' which immediately processes it.
6. $\texttt{start-corrupt}(\mathsf{ID})$: from now on the entire internal state and randomness of ID is leaked to the adversary.
7. $\texttt{end-corrupt}(\mathsf{ID})$: ends the leakage of user ID's internal state and randomness to the adversary.
8. $\texttt{challenge}(q^*)$: A picks a query q^* corresponding to an action $\mathsf{a}^* \in \{\texttt{add}, \texttt{remove}, \texttt{update}\}$ or the initialization (if $q^* = 1$). Let $\texttt{K}^0$ denote the group key that is sampled during this operation and $\texttt{K}^1$ be a fresh random key. The challenger tosses a coin b and – if the safe predicate below is satisfied – the key $\texttt{K}^b$ is given to the adversary (if the predicate is not satisfied the adversary gets nothing).

At the end of the game, the adversary outputs a bit b' and wins if $b' = b$. We call a CGKA scheme (Q, M, t, ϵ)-CGKA-secure if for any adversary A making at most Q queries of the form $\texttt{add}(\cdot,\cdot)$, $\texttt{remove}(\cdot,\cdot)$, or $\texttt{update}(\cdot)$ on a group of size[9] at most M and running in time t it holds

$$\mathsf{Adv}_{\mathsf{CGKA}}(\mathsf{A}) := |\mathsf{Pr}[1 \leftarrow \mathsf{A}|b = 0] - \mathsf{Pr}[1 \leftarrow \mathsf{A}|b = 1]| < \epsilon.$$

5.3.2 The Safe Predicate

We define the *safe predicate* to rule out trivial winning strategies and at the same time restrict the adversary as little as possible. For example, if the adversary challenges the first ($\texttt{initialise}$) query and then corrupts a user in the group, they can trivially distinguish the real group key from random. Thus, intuitively, we call a query q^* safe if the group key generated in response to query q^* is not computable from any compromised state. Since each group key is encrypted to at most one init key for each party, this means that the users which are group members[10] at time q^* must not be compromised as long as these init keys are part of their state. However, defining a reasonable safe predicate in terms of allowed sequences of actions is very subtle.

To gain some intuition, consider the case where query q^* is an update for a party ID^*. Then, clearly, ID^* must not be compromised right after it generated the update. On the other

[9] Note that we consider an asynchronous setting were users might have different view on the current state of the group and, even worse, in our partially active security model A might force users into inconsistent states. Thus, the size of the group at some time point q is not well-defined. To be precise, M is an upper bound on the number of group members in the view of *any* user.

[10] To be precise, since parties might be in inconsistent states, group membership is not unique but rather depends on the users' *views* on the group state. We will discuss this below.

hand, since the update function was introduced to heal a user's state and allow for PCS, any corruption of ID* *before* q^* should not harm security. Similarly, any corruption of ID* *after* a further processed update operation for ID* should not help the adversary either (compare FS). Finally, also in the case where the update generated at time q^* is rejected to ID* and ID* processes this message of the form $\texttt{confirm}(q^*, 0)$ by returning to its previous state, any corruption of ID* after processing the reject message should not affect security of the challenge group key. All these cases should be considered safe.

Additionally, we have to take care of other users which are part of the group when the challenge key is generated: For a challenge to be safe, we must make sure that the challenge group key is not encrypted to any compromised key. At the same time, one has to be aware of the fact that in the asynchronous setting the view of different users might differ substantially. As mentioned above, we consider inconsistency of users' states rather a matter of functionality than security, and aim to define the safe predicate as unrestrictive as possible, to also guarantee security for inconsistent group states. For example, consider the following scenario: user ID generates an update during an uncompromised time period and processes a reject for this update still in the uncompromised time period, but this update is confirmed to and processed by user ID* before she does her challenge update q^*; this results in a safe challenge, since the challenge group key is only encrypted to the new init key, which is not part of ID's state at any compromised time point. However, one has to be careful here, since in a similar scenario where ID does not process the reject for its own update, the challenge group key would clearly not be safe anymore.

For the following definitions we consider discrete time steps measured in terms of the number of queries that have been issued by the adversary so far.

We first identify for each user a critical window in the view of a specific user ID*. The idea is to define exactly the time frame in which a user may leak a group key if ID* generates it at a specific point in time and distributes it to the group. Clearly, the users may not be corrupted in this time frame if this happens to be the challenge group key.

Definition 37 (Critical window, safe user)**.** Let ID and ID* be two (not necessarily different) users and $q^* \in [1, Q]$ be some query. Let $q^- \leq q^*$ be the query that set ID's current key *in the view of* ID* *at time* q^*, i.e. the query $q^- \leq q^*$ that corresponds to the last update message $\mathsf{a}^-_{\mathsf{ID}} := \texttt{update}(\mathsf{ID})$ processed by ID* at some time point $[q^-, q^*]$ (see Figure 5.3). If ID* does not process such a query then we set $q^- = 1$, the first query. Analogously, let $q^+ \geq q^-$ be the first query that invalidates ID's current key, i.e. ID processes one of the following two confirmations:

1. $\texttt{confirm}(\mathsf{a}^-_{\mathsf{ID}}, 0)$, the rejection of action $\mathsf{a}^-_{\mathsf{ID}}$; or
2. $\texttt{confirm}(\mathsf{a}^+_{\mathsf{ID}}, 1)$, the confirmation of an update $\mathsf{a}^+_{\mathsf{ID}} := \texttt{update}(\mathsf{ID}) \neq \mathsf{a}^-_{\mathsf{ID}}$.

If ID does not process any such query then we set $q^+ = Q$, the last query. We say that the window $[q^-, q^+]$ is *critical* for ID at time q^* in the view of ID*. Moreover, if the user ID is *not corrupted* at any time point in the critical window, we say that ID is *safe* at time q^* in the view of ID*.

We are now ready to define when a *group key* should be considered *safe*. The group key is considered to be safe if all the users that ID* considers to be in the group are individually safe, i.e., not corrupted in its critical window, in the view of ID*. We point out that there is an exception when the action that generated the group key $\mathtt{K}^*$ is a self-update by ID* where, to *allow healing*, instead of the normal critical window we use the window $[q^*, q^+]$ as critical.

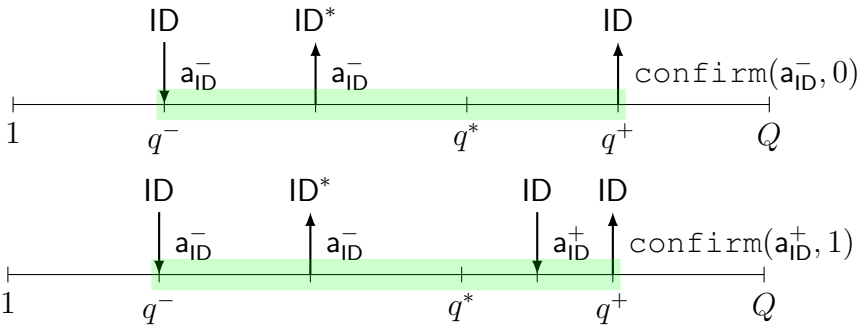


Figure 5.3: A schematic diagram showing the critical window for a user ID in the view of another user ID* with respect to query q^*. An arrow from a user to the timeline is interpreted as a request by the user, whereas an arrow in the opposite direction is interpreted as the user processing the message. The figure at top (resp., bottom) corresponds to the first (resp., second) case in Definition 37.

Definition 38 (Safe predicate)**.** Let $\mathtt{K}^*$ be a group key generated in an action

$$\mathsf{a}^* \in \{\texttt{add}(\mathsf{ID}^*, \cdot), \texttt{remove}(\mathsf{ID}^*, \cdot), \texttt{update}(\mathsf{ID}^*), \texttt{initialise}(\mathsf{ID}^*, \cdot)\}$$

at time point $q^* \in [1, Q]$ and let $\mathcal{U}^*$ be the group of users which would end up in the group if query q^* was processed, as viewed by the generating user ID*. Then the key $\mathtt{K}^*$ is considered *safe* if for all users $\mathsf{ID} \in \mathcal{U}^*$ (including ID*) we have that ID is safe at time q^* in the view of ID* (as per Definition 37) with the following exceptional case: if $\mathsf{ID} = \mathsf{ID}^*$ and $\mathsf{a}^* = \texttt{update}(\mathsf{ID}^*)$ then we require ID* to be safe w.r.t. the window $[q^*, q^+]$.

5.3.3 The Challenge Graph

In the last section, we defined what it means for a group key to be safe via a *safe predicate*. In this section, we try to interpret the safe predicate for the TTKEM protocol. That is, our goal is to show that if the safe predicate is satisfied for a group key $\mathtt{K}^* = \Delta^*$ generated while playing the CGKA game on TTKEM, then none of the seeds or secret keys *used to derive* this group key are leaked to the adversary (Lemma 21) – this fact will be crucial in Section 5.3.5, where we argue the security of TTKEM using the Piecewise-Guessing framework presented in Chapter 3. To this end, we view the CGKA game for TTKEM as a game on a graph and then define the *challenge graph* for challenge group key $\mathtt{K}^*$ as a sub-graph of the whole CGKA graph.

The CGKA graph. A node i in the CGKA graph for TTKEM is associated with seeds Δ_i and $s_i := H_2(\Delta_i)$ and a key-pair $(\mathtt{pk}_i, \mathtt{sk}_i) := \mathsf{Gen}(s_i)$ (as defined in Equation 5.1). The edges of the graph, on the other hand, are induced by dependencies via the hash function H_1 or (public-key) encryptions. To be more precise, an edge (i, j) might correspond to either:

1. a ciphertext of the form $\mathsf{Enc}_{\mathtt{pk}_i}(\Delta_j)$; or
2. an application of H_1 of the form $\Delta_j = H_1(\Delta_i)$ used in hierarchical derivation.

Naturally, the structure of the CGKA graph depends on the `update`, `add` or `remove` queries made by the adversary, and is therefore generated adaptively.

The challenge graph. The challenge graph for $\mathtt{K}^* = \Delta^*$, intuitively, is the sub-graph of the CGKA graph induced on the nodes from which Δ^* is trivially derivable. Therefore, according to the definition of the CGKA graph, this consists of nodes from which Δ^* is reachable and the corresponding edges (used to reach Δ^*). For instance, in the case where the adversary maintains all users in a consistent state and there are no tainted nodes, the challenge graph would simply be the binary tree rooted at Δ^* with leaves corresponding to init keys of users in the group at that point. When the group view is inconsistent among the users these leaves would correspond to the init keys of users in the view of ID^*. Moreover, if there are tainted nodes, the tree could also have (non-init key) leaves corresponding to these tainted nodes. Below we state and prove the key lemma which connects the safe predicate to the challenge graph of TTKEM.

Lemma 21. *For any* safe *challenge group key in TTKEM it holds that none of the seeds and secret keys in the challenge graph is leaked to the adversary via corruption.*

Proof. We first observe that in the CGKA game, when a user is corrupted all secret keys and seeds in its memory are also corrupt and therefore leaked to the adversary. This could consist of the secret keys and seeds on the *current* path from the user's leaf to the root (in the tree as currently viewed by the user itself), secret keys on a *pending* path (if one exists) and, by definition, all tainted nodes (as viewed by any user). Let's consider the challenge graph of any group key Δ^* generated by a user ID^* during an action

$$\mathsf{a}^* \in \{\mathtt{add}, \mathtt{remove}, \mathtt{update}, \mathtt{initialise}\}$$

at a time point q^*. Intuitively, the safe predicate for TTKEM ensures that *none* of the keys in this graph is leaked to the adversary by requiring that every node has been refreshed (via an update) before the generation of Δ^*. Therefore, information on Δ^* (via ciphertexts or hash dependencies) are sent *only* to refreshed and therefore uncorrupt nodes. Since the rest of the simulation in the CGKA game is independent of Δ^*, an adversary learns nothing about it. We formalise this intuition below via a proof by contradiction: assuming Δ^* is leaked to the adversary, we show that the safe predicate for Δ^* is somehow violated in the view of ID^* at q^*.

Let's first consider a restricted adversary that does not issue reject queries (i.e, $\mathtt{confirm}(\cdot, 0)$). The fact that Δ^* is leaked to the adversary means that some secret key $\mathtt{sk}$ or a seed s or Δ in the challenge graph was leaked during the CGKA game; assume some key $\mathtt{sk}$ was leaked (the argument when a seed s or Δ is leaked is similar). This in turn means that some user was corrupted while having $\mathtt{sk}$ in its memory. There are two possibilities: either $\mathtt{sk}$ corresponds to a leaf node and is a user key for a user ID (possibly same as ID^*) which is in the group in the view of ID^* at time q^* (we argue on that in the end of the proof) (Case A) or $\mathtt{sk}$ corresponds to an internal node (Case B). Let's argue these cases separately.

Case A. Let's suppose first that $\mathsf{ID} \neq \mathsf{ID}^*$. As $\mathtt{sk}$ is leaked to the adversary it follows that ID was corrupted *after* ID generated the update ($\mathsf{a}^-_{\mathsf{ID}}$) and *before* ID processed the update $\mathsf{a}^+_{\mathsf{ID}} \neq \mathsf{a}^-_{\mathsf{ID}}$ (see Figure 5.4, top). The reason being that this is the *only* window in which this key is present in the memory of ID (and this key is *not* present in the memory of any other user). However, this violates the safe predicate (restricted to the case where there are no rejects) as ID is not a safe user. In the complementary case where $\mathsf{ID} = \mathsf{ID}^*$, the argument is similar except that the window in which $\mathtt{sk}$ is present in the memory is different (as in the exceptional case in Definition 38).

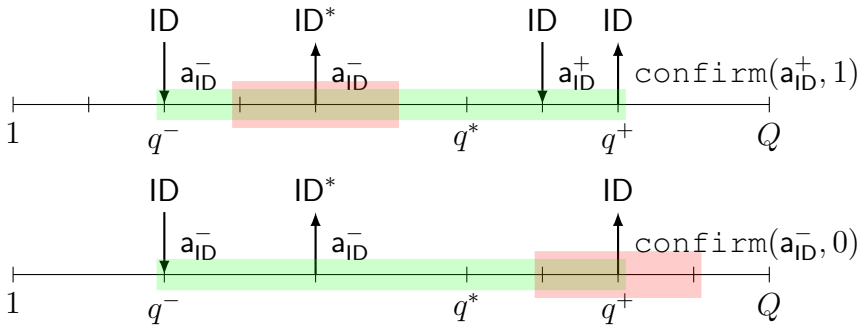


Figure 5.4: Violation of the safe predicate. The critical window for ID is shaded in green, and the windows where ID was corrupt is shaded in red. In all the cases, there is an intersection between the critical and corrupt window and therefore the safe predicate is violated.

Case B. When $\mathtt{sk}$ is part of an internal node, we apply the argument in Case A to every user that is an ancestor of (the node) $\mathtt{sk}$ in the challenge graph: it would then follow that at least one of these users violates the safe predicate. However, if not every leaf in the set of ancestors of (the node) $\mathtt{sk}$ in the challenge graph corresponds to a particular user, which can *only* happen when it corresponds to a tainted node, we apply this argument to the owner of the tainted node. Note that, by construction, the owner must be part of the group as viewed by ID* (even if their states are inconsistent at time q^*) as all tainted nodes corresponding to a removed user are also removed from the state.

Now, in the case where the adversary does reject queries, the argument is similar except that we have to take into account the case where a user key $\mathtt{sk}$ is invalidated by such rejects. To be precise, in the case where $\mathsf{a}^-_{\mathsf{ID}}$ is rejected to ID it could be the case that the safe predicate is violated when a corruption occurs before $\mathsf{a}^-_{\mathsf{ID}}$ is rejected to ID (see Figure 5.4, bottom). Finally, note that in the above argument it was implicitly assumed that all users corresponding to the leaf nodes of Δ^* were part of the group $\mathcal{U}^*$ viewed by ID* at time q^*. This is guaranteed by the fact that users only process messages if the corresponding transcript hash coincides with the hash in their current state, which in particular guarantees that the view of the group coincides among a user generating a message and a user processing that message, i.e. ID* is aware of all edges in the challenge tree of Δ^*. This is crucial since the safe predicate is applied *only* to the users in $\mathcal{U}^*$. □

5.3.4 Security Proof for TTKEM in the Standard Model

To prove security of TTKEM in the standard model, we will use the Piecewise-Guessing framework (Chapter 3). Recall that in the CGKA security game, the aim of the adversary is to distinguish a safe challenge group key $\mathtt{K}^* = \Delta^*$ from a uniformly random and independent seed. We first consider the *selective* CGKA game, where the adversary has to do all its queries at once. We call the two possible executions of the game the *real* and *random* CGKA game and aim to proof indistiguishability of these two games via a sequence of indistinguishable hybrid games. Similar to several other applications of the framework in Chapter 3, we will define these hybrid games via the so-called *reversible black pebbling* game, introduced by Bennett [Ben89], where, given a directed acyclic graph with unique sink (here, the challenge graph), in each step one can put or remove one pebble on a node following certain rules, and the goal is to reach the pebbling configuration where there is only one pebble on the sink of the graph. Each *pebbling configuration* $\mathcal{P}_\ell$ then uniquely defines a *hybrid game* Hyb_ℓ: a node v in the

tree being pebbled means that in this hybrid game whenever Δ_v would be used to answer a query, a freshly chosen random seed (independent of Δ_v) is used instead in the simulation. This applies to all cases where Δ_v would be used as input for H_1 or H_2, or as the challenge output (if v is the challenge node). All remaining nodes and edges are simulated as in the real CGKA game. Thus, the real game $\mathsf{Hyb}_{\mathsf{real}}$ is represented as the empty pebbling configuration $\mathcal{P}_0$ where there is no pebble at all, while the random game $\mathsf{Hyb}_{\mathsf{random}}$ corresponds to the final configuration $\mathcal{P}_\tau$ where only the sink node is pebbled (where τ denotes the length of the pebbling sequence).

Definition 39 (Reversible black pebbling)**.** A reversible pebbling of a directed acyclic graph $G = (V, E)$ with unique sink T is a sequence $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ with $\mathcal{P}_\ell \subseteq V$ $(\ell \in [0, \tau])$, such that $\mathcal{P}_0 = \emptyset$ and $\mathcal{P}_\tau = \{T\}$, and for all $\ell \in [1, \tau]$ there is a unique $v \in V$ such that:

- $\mathcal{P}_\ell = \mathcal{P}_{\ell-1} \cup \{v\}$ or $\mathcal{P}_\ell = \mathcal{P}_{\ell-1} \setminus \{v\}$,
- for all $u \in \mathrm{parents}(v)$: $u \in \mathcal{P}_{\ell-1}$.

By Lemma 21, we know that none of the seeds or secret keys in the challenge graph is leaked to the adversary throughout the entire game. This will allow us to prove indistinguishability of subsequent hybrid games from IND-CPA security of the underlying encryption scheme and pseudorandomness of the hash functions H_1, H_2. Recall, the functions H_1, H_2 were defined by a hash function H which takes some Δ_i as secret key and publicly known fixed strings x_1, x_2 as inputs. To guarantee security, H is assumed to be a pseudorandom function, where we will use the following non-standard but equivalent (to the standard) definition of pseudorandomness:

Definition 40 (Pseudorandom function, alternative definition)**.** Let $H : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a keyed function. We define the following game $\mathsf{PRF}(\lambda)$: First, a key $k \leftarrow \{0,1\}^\lambda$ is chosen uniformly at random and the adversary is given access to an oracle $H(k, \cdot)$. When the adversary outputs a string $x \leftarrow \{0,1\}^\lambda$, a uniformly random bit $b \leftarrow \{0,1\}$ is chosen and the adversary receives either $H(k, x)$ in the case $b = 0$, or $y \in \{0,1\}^\lambda$ uniformly at random if $b = 1$. Finally, the adversary outputs a bit b'. If x was never queried to the oracle $H(k, \cdot)$ and $b' = b$, then the output of the game is 1, otherwise 0. We call H (t, ϵ)-*pseudorandom* if for all adversaries A running in time t we have

$$\mathsf{Adv}_{\mathsf{PRF}}(\mathsf{A}) := |\mathsf{Pr}[1 \leftarrow \mathsf{PRF}(\lambda) | b = 0] - \mathsf{Pr}[1 \leftarrow \mathsf{PRF}(\lambda) | b = 1]| < \epsilon.$$

It is an easy exercise to prove that the above definition is equivalent to the standard textbook definition of pseudorandom functions (i.e., only a polynomial loss in security is involved by the respective reductions).

Lemma 22. *Let $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ be a valid pebbling sequence on the challenge graph. If H is an (t, ϵ)-secure pseudorandom function and $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a (t, ϵ)-IND-CPA secure encryption scheme, then any two subsequent hybrid games $\mathsf{Hyb}_\ell, \mathsf{Hyb}_{\ell+1}$ are $(t, 5 \cdot \epsilon)$-indistinguishable*[11]*.*

Proof. Let $\mathsf{Hyb}_\ell, \mathsf{Hyb}_{\ell+1}$ be two subsequent hybrid games. We assume that $\mathcal{P}_{\ell+1}$ differs from $\mathcal{P}_\ell$ by one additional pebble on a node v^* with ingoing encryption edge (u, v^*) and

[11] Technically, the t in Lemma 22 changes slightly due to the reduction and thus should not actually be the same t. For simplicity, in all our security reductions we will ignore such miniscule running time overheads incurred by simulating challengers of the security games or sampling (a small number of) random bits.

ingoing H_1 edge (u', v^*). The case where $\mathcal{P}_{\ell+1}$ is obtained from $\mathcal{P}_\ell$ by removing one pebble can be proven in a similar way. We proof indistinguishability by a sequence of hybrids $\mathsf{Hyb}_\ell := \mathsf{Hyb}_{\ell,0}, \mathsf{Hyb}_{\ell,1}, \ldots, \mathsf{Hyb}_{\ell,5} := \mathsf{Hyb}_{\ell+1}$, where the intermediate hybrids are defined as follows:

- $\mathsf{Hyb}_{\ell,1}$ is defined similarly to $\mathsf{Hyb}_{\ell,0}$ except that the key pair $(\mathtt{pk}_u, \mathtt{sk}_u)$ is generated from a uniformly random seed instead of the output s_u of H_2.
- $\mathsf{Hyb}_{\ell,2}$ is defined similarly to $\mathsf{Hyb}_{\ell,1}$ except that the encryption $\mathsf{Enc}_{\mathtt{pk}_u}(\Delta_{v^*})$ is replaced by an encryption of a uniformly random seed.
- $\mathsf{Hyb}_{\ell,3}$ is defined similarly to $\mathsf{Hyb}_{\ell,2}$ except that instead of Δ_{v^*} a uniformly random seed is used as input to H_1, or H_2, or the challenge output (if v^* is the challenge node) whenever needed to answer any queries of the adversary.
- $\mathsf{Hyb}_{\ell,4}$ is defined similarly to $\mathsf{Hyb}_{\ell,3}$ except that the encryption of a uniformly random seed is replaced by $\mathsf{Enc}_{\mathtt{pk}_u}(\Delta_{v^*})$.
- $\mathsf{Hyb}_{\ell,5}$ is defined similarly to $\mathsf{Hyb}_{\ell,3}$ except that the key pair $(\mathtt{pk}_u, \mathtt{sk}_u)$ is generated from $s_u = H_2(\Delta_u)$ again, instead of using a uniformly random seed. Note that indeed $\mathsf{Hyb}_{\ell,5} = \mathsf{Hyb}_{\ell+1}$ holds.

Indistinguishability of $\mathsf{Hyb}_{\ell,0}$ and $\mathsf{Hyb}_{\ell,1}$ follows from the pseudorandomness of the hash function H: Since by the pebbling rules it must hold that node u is pebbled, in game $\mathsf{Hyb}_{\ell,0}$ the seed s_u for key generation is computed by applying H_2 to a uniformly random seed instead of Δ_u. Thus, by pseudorandomness of H the seed s_u is indistinguishable from a uniformly random seed, as used in $\mathsf{Hyb}_{\ell,1}$. Furthermore, all the remaining seeds and edges needed during simulation of the hybrid games can be perfectly simulated, which implies that any advantage ϵ of an adversary in distinguishing $\mathsf{Hyb}_{\ell,0}$ from $\mathsf{Hyb}_{\ell,1}$ leads to the same advantage ϵ in distinguishing H from a random function.

Indistinguishability of $\mathsf{Hyb}_{\ell,1}$ and $\mathsf{Hyb}_{\ell,2}$ follows from the IND-CPA security of the encryption scheme: Since in game $\mathsf{Hyb}_{\ell,1}$ the key pair $(\mathtt{pk}_u, \mathtt{sk}_u)$ is generated just as in Gen, a reduction can embed an IND-CPA challenge for messages Δ_{v^*} and a uniformly random seed at the encryption edge (u, v^*) and otherwise perfectly simulate all answers to queries of the adversary. Thus, any adversary distinguishing $\mathsf{Hyb}_{\ell,1}$ from $\mathsf{Hyb}_{\ell,2}$ with advantage ϵ can be used to break IND-CPA security of the encryption scheme with the same advantage.

Indistinguishability of $\mathsf{Hyb}_{\ell,2}$ and $\mathsf{Hyb}_{\ell,3}$ follows from the pseudorandomness of the hash function H: This is similar to the case of $\mathsf{Hyb}_{\ell,0}$ and $\mathsf{Hyb}_{\ell,1}$, and the indistinguishability of Δ_{v^*} again cruicially relies on the fact that node u' is pebbled. However, the alternative definition of pseudorandomness must be used for the reduction, since the seed $s_{u'}$ used to generate the key pair $(\mathtt{pk}_{u'}, \mathtt{sk}_{u'})$ is the output of H with the same random seed as used to compute Δ_{v^*}. Again, the remaining simulation can be done perfectly and the reduction preserves the advantage.

Similar to the case of $\mathsf{Hyb}_{\ell,1}$ and $\mathsf{Hyb}_{\ell,2}$, indistinguishability of $\mathsf{Hyb}_{\ell,3}$ and $\mathsf{Hyb}_{\ell,4}$ follows from the IND-CPA security of the encryption scheme.

Finally, indistinguishability of $\mathsf{Hyb}_{\ell,4}$ and $\mathsf{Hyb}_{\ell,5}$ follows from pseudorandomness of the hash function H, similar to the case of $\mathsf{Hyb}_{\ell,0}$ and $\mathsf{Hyb}_{\ell,1}$.

Note that in all the sketched reductions, the running time remains essentially the same. Thus, the claim follows. □

Choosing a trivial pebbling sequence of the challenge graph, this already implies *selective* CGKA security of TTKEM. Unfortunately, in the adaptive setting, the challenge graph is not known to the reduction until the adversary does its challenge query, but by this time it will be too late for the reduction to embed a challenge, since seeds and public keys in the challenge graph might have been used already before when answering previous queries by the adversary. Thus, to simulate a hybrid game Hyb_ℓ, the reduction needs to *guess* (some of) the adaptive choices the adversary will do. Naïvely, this would result in an exponential security loss. However, the Piecewise-Guessing framework from Chapter 3 allows to do significantly better:

The problem of proving CGKA security of TTKEM now reduces to finding a sequence of indistinguishable hybrids such that each hybrid can be simulated by only a small amount of random guessing. Defining hybrid games via pebbling configurations as above and using the space-optimal pebbling sequence for binary trees, described in Algorithm 4.8 in Section 4.3.2, which uses $\tau = M^2$ steps and only $2\log(M) + 1$ pebbles[12] (where M denotes an upper bound on the number of users in any group), implies a security reduction for TTKEM with only a quasipolynomial loss in security.

Theorem 16. *If H is an (t, ϵ)-pseudorandom function and $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is an (t, ϵ)-IND-CPA secure encryption scheme, then TTKEM is $(Q, M, t, 5 \cdot M^2 \cdot Q^{\log(M)+2} \cdot \epsilon)$-CGKA secure.*

Proof. Note that the challenge graph is a complete binary tree of depth $\log(M)$ in the worst case and let $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ be the recursive pebbling strategy for binary trees from Algorithm 4.8, which uses $\tau = M^2$ steps and at most $2\log(M) + 1$ pebbles. We will prove that each pebbling configuration $\mathcal{P}_\ell$ can be represented using $(\log(M) + 2) \cdot \log Q$ bits. The claim then follows by Lemma 22 and Theorem 2.

We need the following property of the strategy $\boldsymbol{\mathcal{P}}$: For all $\ell \in [0, \tau]$, there exists a leaf in the tree such that all pebbled nodes lie either on the path from that leaf to the sink or on the copath. Furthermore, the subgraph on this set of potentially pebbled nodes contains $2\log(M) + 1$ nodes which are connected by at most $\log(M) + 1$ encryption and H_1 edges, respectively. Throughout the game, the reduction always knows in which position in the binary tree a node ends up, but it does not know which of the up to Q versions of the node will end up in the challenge tree. However, nodes connected by an H_1 edge are generated at the same time, so the reduction only needs to guess for at most $\log(M) + 2$ nodes which of the up to Q versions of that node will be in the challenge graph. This proves the claim. □

Since the above proof mainly relies on the *depth* of the challenge tree, it can easily be adapted to prove CGKA security of TreeKEM, the main difference being the different challenge graph structure (i.e. higher indegree) induced by blanking.

[12] Although the original Lemma 8 in Section 4.3.2 states that $3\log(M)$ pebbles are required to pebble a binary tree, the bound is loose since it is derived from Lemma 7. It is not difficult to see that a tighter analysis of Algorithm 1 for the case of binary trees leads to a bound of $2\log M + 1$.

5.3.5 Security Proof for TTKEM in the Random Oracle Model

The security of TTKEM is closely related to the notion of generalized selective decryption (GSD), which we adapt to the public key setting for our purposes:

Definition 41 (Generalized selective decryption (GSD), adapted to public-key setting)**.** Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme with secret key space $\mathcal{K}$ and message space $\mathcal{M}$ such that $\mathcal{K} \subseteq \mathcal{U}$. The *GSD game* (for public key encryption schemes) is a two-party game between a challenger C and an adversary A. On input an integer N, for each $v \in [1, N]$ the challenger C picks a key pair $(\mathtt{pk}_v, \mathtt{sk}_v) \leftarrow \mathsf{Gen}(r)$ (where r is a random seed) and initializes the *key graph* $G = (\mathcal{V}, \mathcal{E}) := ([1, N], \emptyset)$ and the set of corrupt users $\mathcal{C} = \emptyset$. A can adaptively do the following queries:

- $(\mathtt{encrypt}, u, v)$: On input two nodes u and v, C returns an encryption $\mathtt{c} = \mathsf{Enc}_{\mathtt{pk}_u}(\mathtt{sk}_v)$ of $\mathtt{sk}_v$ under $\mathtt{pk}_u$ along with $\mathtt{pk}_u$ and adds the directed edge (u, v) to $\mathcal{E}$. Each pair (u, v) can only be queried at most once.
- $(\mathtt{corrupt}, v)$: On input a node v, C returns $\mathtt{sk}_v$ and adds v to $\mathcal{C}$.
- $(\mathtt{challenge}, v)$, single access: On input a challenge node v, C samples $b \leftarrow \{0, 1\}$ uniformly at random and returns $\mathtt{sk}_v$ if $b = 0$, otherwise it returns a new secret key generated by Gen using a new independent uniformly random seed. In the context of GSD we denote the *challenge graph* as the graph induced by all nodes from which the challenge node v is reachable. We require that none of the nodes in the challenge graph are in $\mathcal{C}$, that G is acyclic and that the challenge node v is a sink. Note that A does not receive the public key of the challenge node, since it is a sink.

Finally, A outputs a bit b' and it *wins* the game if $b' = b$. We call the encryption scheme (t, ϵ)-adaptive GSD-secure if for any adversary A running in time t it holds

$$\mathsf{Adv}_{\mathsf{GSD}}(\mathsf{A}) := |\mathsf{Pr}[1 \leftarrow \mathsf{A} | b = 0] - \mathsf{Pr}[1 \leftarrow \mathsf{A} | b = 1]| < \epsilon.$$

We first prove a general result for our version of GSD, which could be of independent interest.

Theorem 17. *For any public key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *and hash function* H *let the encryption scheme* $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ *be defined as follows: 1)* Gen' *simply picks a random seed* s *as secret key and runs* $\mathsf{Gen}(H(s))$ *to obtain the corresponding public key, 2)* Enc' *is identical to* Enc *and 3)* Dec'*, given the secret key* s*, extracts the secret key from* $\mathsf{Gen}(H(s))$ *and uses* Dec *to decrypt the ciphertext.*

If Π *is* $(t, \tilde{\epsilon})$*-IND-CPA secure and* H *is modelled as a random oracle, then* Π' *is* (t, ϵ)*-adaptive GSD secure, where* $\epsilon = 3N^2 \cdot \tilde{\epsilon} + \frac{3mN}{2^\lambda}$*, with* N *the number of nodes,* m *the number of oracle queries to* H*,* λ *the seed length.*

Proof. Note that the GSD graph contains nodes $u \in [1, N]$ corresponding to seeds s_u that are hashed using the RO H to obtain seeds for the encryption scheme Π. We prove GSD security by a sequence of hybrids interpolating between the *real* game GSD_0 where the challenge query v is answered with the real seed s_v and the *random* game GSD_1 where it is answered with an independent uniformly random seed in $\mathcal{S}$, where $\mathcal{S}$ denotes the seed space.

- Define $\mathsf{G}_0 := \mathsf{GSD}_0$, the real GSD game.

- Let $s' \in \mathcal{S}$ and v be the challenge node. For $1 \leq i \leq \delta_{in}(v)$ we define the hybrid game G_i as follows: The game is similar to G_{i-1} except that the ith query of the form $(\texttt{encrypt}, u, v)$ is answered by $\mathsf{Enc}_{\mathrm{pk}_u}(s')$.

Note, the game $\mathsf{G}_{\delta_{in}(v)}$ is distributed exactly the same as GSD_1. Thus, in this case, for any GSD-adversary A with advantage ϵ, the advantages of A in distinguishing hybrid games G_{i-1} from G_i sum up to at least ϵ. Since two subsequent hybrid games differ in exactly one encryption edge, we will use this distinguishing advantage to solve an IND-CPA challenge. To simulate game G_i, the reduction simply *guesses* the challenge node v as well as the source node u of the ith encryption incident on v. We denote these guesses by v^* and u^*. This simulation, however, is only possible if A does not query its oracle H on any of the seeds corresponding to the parents of $v^* = v$, since otherwise A can trivially distinguish G_0 from $\mathsf{G}_{\delta_{in}(v)}$. We encompass this issue by the following (more general) event.

- Event E: A queried a seed s^* to the random oracle which corresponds to a node that was not corrupted and is not reachable by any corrupted node, and no challenge query was issued for it.

Intuitively, event E is true if A queried the random oracle H on some seed which it doesn't trivially know and which is associated with a node that might end up in the challenge graph.

We make a case distinction on the probability $\gamma_0 = \Pr[\text{A triggers } E \mid \mathsf{GSD}_0]$ is the probability that A triggers E when playing the GSD_0 game: If $\gamma_0 < \epsilon/2$, apply Lemma 54, which results in an advantage $> \frac{\epsilon}{2N^2}$ against IND-CPA security. Else, apply Corollary 21 to obtain an IND-CPA adversary with advantage $> \frac{\epsilon}{2N^2} - \frac{m}{N2^\lambda}$.

In either case we obtain an adversary with advantage $> \frac{\epsilon}{2N^2} - \frac{m}{N2^\lambda}$ against IND-CPA security, which proves the claim.[13] □

For ease of exposition, we prove a slightly weaker result in the main body of this work and refer to the appendix (Section A.1) for a more involved optimized proof. The proof of Lemma 23 – simplifying Lemma 54 in the appendix – relies on the Piecewise-Guessing framework (Cha). The framework allows for a more modular reduction and we believe that the proof of Lemma 23 already gives a good intuition on the final reduction presented in Lemma 54. This approach involves an additional loss in the indegree of the key graph. While the indegree is a small constant for the case of TTKEM, it can be up to linear in n, the number of users, for TreeKEM. Thus, the optimized proof of Theorem 17 is of particular interest if applied to TreeKEM, similar to Theorem 18. We mention that Panjwani [Pan07] already gave a very similar proof for (private-key) GSD on graphs of bounded depth and the result we obtain in Lemma 54 is similar to Panjwani's result for the special case of graphs of depth 1. This is not a coincidence, since as long as the adversary does not query a seed in the challenge graph, the random oracle ensures that the adverdary is effectively playing a GSD game on a depth 1 graph. Since Panjwani's proof is very long and technical, we provide a self-contained proof in the appendix, which is precisely adapted to our setting of *public-key GSD in the random oracle model*. In the following, by the GSD game we refer to the public-key GSD game on the encryption scheme Π from Theorem 17.

We first consider the case of GSD adversaries which trigger event E only with small probability.

[13] Note, this reduction is *non-uniform*. One can get a uniform reduction by combining these two reductions and guessing whether event E will happen.

Lemma 23. *Let the event E be defined as in the proof of Theorem 17 and* A *an adversary against GSD such that*

- $\Pr[\mathsf{A} \to 0 \mid \mathsf{GSD}_0] - \Pr[\mathsf{A} \to 0 \mid \mathsf{GSD}_1] = \epsilon$ *and*
- $\gamma_0 \leq \frac{\epsilon}{2}$ *where* $\gamma_b = \Pr[\mathsf{A}$ *triggers* $E \mid \mathsf{GSD}_b]$.

Then, in the random oracle model, there exists an IND-CPA adversary A′ *(that has essentially the same running time) against the underlying encryption scheme with advantage* $> \frac{\epsilon}{2\delta_{in}N^2}$, *where N is the number of nodes in the key graph and δ_{in} is an upper bound on the maximium indegree of the key graph queried by* A.

Proof. We first consider a variant GSD′ of GSD where the game aborts as soon as E happens (we consider this as A outputting 1, i.e. no advantage can be gained on this event). We have

$$\begin{aligned}
\epsilon &= \Pr[\mathsf{A} \to 0 \mid \mathsf{GSD}_0] - \Pr[\mathsf{A} \to 0 \mid \mathsf{GSD}_1] \\
&= \Pr[(\mathsf{A} \to 0) \wedge E \mid \mathsf{GSD}_0] - \Pr[(\mathsf{A} \to 0) \wedge E \mid \mathsf{GSD}_1] \\
&\quad + \Pr[(\mathsf{A} \to 0) \wedge \neg E \mid \mathsf{GSD}_0] - \Pr[(\mathsf{A} \to 0) \wedge \neg E \mid \mathsf{GSD}_1] \\
&= \Pr[\mathsf{A} \to 0 \mid E \wedge \mathsf{GSD}_0] \cdot \gamma_0 - \Pr[\mathsf{A} \to 0 \mid E \wedge \mathsf{GSD}_1] \cdot \gamma_1 \\
&\quad + \Pr[(\mathsf{A} \to 0) \wedge \neg E \mid \mathsf{GSD}_0] - \Pr[(\mathsf{A} \to 0) \wedge \neg E \mid \mathsf{GSD}_1] \\
&\leq \frac{\epsilon}{2} + \Pr[(\mathsf{A} \to 0) \wedge \neg E \mid \mathsf{GSD}_0] - \Pr[(\mathsf{A} \to 0) \wedge \neg E \mid \mathsf{GSD}_1]
\end{aligned}$$

So clearly A has advantage $> \epsilon/2$ in GSD′. We now show that GSD'_0 and GSD'_1 are indistinguishable. We will do so by applying the piecewise guessing technique from Theorem 2. To this aim, first consider the *selective* version of the GSD′ game, where the adversary has to commit to all its queries in the beginning of the game, i.e., the entire graph structure, the challenge v, as well as the subset of corrupted nodes are known to the reduction before executing the GSD game. Let H_i denote the selective version (with $v^* = v$) of the game G_i for $i \in [1, \delta_{in}(v^*)]$.

From an adversary that distinguishes two subsequent games H_{i-1} and H_i one can easily construct an adversary A′ with the same advantage against the IND-CPA game: A′ chooses two random seeds s, s' as its messages and receives a challenge ciphertext c in return. It then simulates the GSD′ game to A and embeds the challenge c in the edge where H_{i-1} and H_i differ. Denote this edge by (u, v). Since A′ generates almost all of the remaining information of the hybrid itself, it can respond faithfully to any query of the adversary. The only tricky queries are 1) a corruption of u and 2) encryption queries of the form (i, u), since this requires to respond with (an encryption of) a seed s_u such that $H(s_u) = r_u$ and applying the key generation algorithm to r_u results in the public key associated to u. Luckily, since u is a parent of the challenge v, u may not be corrupted nor reachable from a corrupted node, so querying H for s_u would trigger E and the game aborts. It follows that we can assign a random seed to u, which to A is information-theoretically indistinguishable to using the correct seed. Now clearly A′ has the same advantage as A since if c encrypts s it is simulating H_{i-1}, while otherwise A' is simulating H_i.

In the adaptive setting, A′ can simulate game G_i by only guessing the challenge node (v) and the source (u) of the ith edge incident on v. Thus, it requires $2\log(N)$ bits of information on the adversary's choices. Then by Theorem 2 it follows that the advantage of A' is at least $\frac{\epsilon}{2\delta_{in} \cdot N^2}$. □

The following Lemma shows that any GSD adversary triggering event E can be reduced to an adversary against a partially selective version of GSD where the challenger aborts whenever event E happens.

Lemma 24. *Let the event E be defined as in the proof of Theorem 17. We consider a slightly modified version of GSD: let GSD''_b be defined like GSD_b with the two differences that a) the game will abort if E happens (we will consider this equivalent to the adversary outputting 1) and b) the adversary has to commit to the challenge node at the beginning of the game, i.e. GSD'' is partially selective. Then for any adversary A which triggers event E with probability ϵ in the GSD_0 game on N nodes there exists an adversary A'' (that has essentially the same running time) in the GSD'' game on $N+1$ nodes such that*

$$\Pr[\mathsf{A}'' \to 0 \mid \mathsf{GSD}''_0] - \Pr[\mathsf{A}'' \to 0 \mid \mathsf{GSD}''_1] \geq \frac{\epsilon}{N} - \frac{m}{2^\lambda},$$

where m is the number of oracle queries to H, λ the seed length.

Proof. In the following we consider event E with respect to A playing the normal GSD game. We construct the adversary A'' as follows. First, A'' *guesses* the node v^* which will be associated with the seed which turns E true, i.e., A'' samples $v^* \leftarrow [1, N]$ uniformly at random and issues a challenge query for v^*; let s^* be the response it receives. Then it runs A. If A queries $(\texttt{encrypt}, u, u')$ for $u \neq v^*$, then A'' just forwards this query to the challenger and returns to A whatever it receives in response. For $u = v^*$, on the other hand, A'' issues a query $(\texttt{encrypt}, N+1, u')$. Note that the node $N+1$ is never used by A and it will associate the key $\mathtt{pk}_{N+1}$ it receives to the node v^*. Unless a seed associated with v^* is queried to the random oracle H, this is indistinguishable from the real GSD game and if this event happens then – assuming the guess v^* was correct – this means that event E turned true. Since A'' aims to use the event E to break its own GSD challenge, it is enough that the game simulated to A until E happens is indistinguishable from the real GSD game.

If A queries $(\texttt{corrupt}, u)$, A'' simply forwards this query to the challenger and returns the response it receives. If A issues a corruption or encryption query such that v^* is reachable from a corrupted node, A'' aborts and outputs 1.

When A queries $(\texttt{challenge}, v)$ for $v \neq v^*$, A'' sends $(\texttt{corrupt}, v)$ to the challenger and receives s_v. Then it samples a bit $c \leftarrow \{0,1\}$ uniformly at random, sets $s_0 := s_v$, samples $s_1 \leftarrow \mathcal{S}$ unifomly at random, and returns s_c to A. Note that v must be a sink node, hence this corruption does not affect the validity of v^* as a GSD challenge. If A happens to choose v^* as its challenge, A'' aborts and outputs 1.

If A makes an oracle query $H(s)$, then A'' just forwards this query to the challenger if $s \neq s^*$, otherwise A'' aborts and outputs the bit $b = 0$. If the seed s^* is never queried (and the game GSD'' does not abort), A'' outputs 1.

Let V denote the event that v^* is associated with the seed which turns E true. Let b^* be the GSD'' challenge bit. Now we first note that

$$\Pr[\mathsf{A}'' \to 0 \mid b^* = 1] \leq \frac{m}{2^\lambda}$$

because if $b^* = 1$, s^* is information-theoretically hidden from A. Furthermore,

$$\begin{aligned}
\Pr[\mathsf{A}'' \to 0 \mid b^* = 0] &= \Pr[(\mathsf{A}'' \to 0) \wedge E \mid b^* = 0] && \text{because } \mathsf{A}'' \to 0 \text{ implies } E \\
&= \epsilon \cdot \Pr[\mathsf{A}'' \to 0 \mid E \wedge (b^* = 0)] && \text{because } \Pr[E \mid b^* = 0] = \epsilon \\
&= \epsilon \cdot \Pr[(\mathsf{A}'' \to 0) \wedge V \mid E \wedge (b^* = 0)] && \text{since } \neg V \text{ implies } \mathsf{A}'' \to 1 \\
&= \epsilon \cdot \Pr[\mathsf{A}'' \to 0 \mid V \wedge E \wedge (b^* = 0)] \Pr[V \mid E \wedge (b^* = 0)] \\
&\geq \frac{\epsilon}{N}
\end{aligned}$$

Where the last step follows since v^* was chosen uniformly at random and the simulation is independent of this choice until E happens and/or A'' aborts, and because $V \wedge E \wedge (b^* = 0)$ implies $\mathsf{A}'' \to 0$. In summary, we obtain

$$\Pr[\mathsf{A}'' \to 0 \mid b^* = 0] - \Pr[\mathsf{A}'' \to 0 \mid b^* = 1] \geq \frac{\epsilon}{N} - \frac{m}{2^\lambda}.$$

This proves the claim. □

The following Corollary now gives a reduction for GSD adversaries which trigger event E with large probability. We optimize the result in Corollary 21 in the appendix.

Corollary 4. *Let the event E be defined as in the proof of Theorem 17, and* A *an arbitrary GSD adversary which triggers E with probability ϵ. Then there exists an IND-CPA adversary* A' *(that has essentially the same running time) with advantage*

$$\Pr[\mathsf{A}' \to 0 \mid b^* = 0] - \Pr[\mathsf{A}' \to 0 \mid b^* = 1] \geq \frac{\epsilon}{\delta_{in} N^2} - \frac{m}{\delta_{in} N 2^\lambda}.$$

where N is the number of nodes, m the number of oracle queries to H, λ the seed length, and δ_{in} an upper bound on the maximium indegree of the graph queried by A*.*

Proof. We first apply Lemma 24 to construct an adversary A'' against GSD'' with advantage $\frac{\epsilon}{N} - \frac{m}{2^\lambda}$. Then we apply a very similar proof as for Lemma 23 to construct an IND-CPA adversary: This proof is exactly the same with the only difference being that GSD'' (played on $N + 1$ nodes) is less adaptive, i.e. A commits to the challenge node v at the beginning of the game, so when applying Theorem 2 to switch from the selective to the adaptive version, A' does not need to guess it. It follows that in order to simulate game G_i, A' needs to guess only the source $u \neq v$ of the ith encryption edge incident on v, which consists of $\log N$ many bits. Accordingly, by Theorem 2, the IND-CPA adversary has advantage

$$\frac{1}{\delta_{in} N} \left(\frac{\epsilon}{N} - \frac{m}{2^\lambda} \right).$$

□

We now adapt the above proof to show a polynomial time reduction for TTKEM in the random oracle model. Intuitively, the CGKA graph corresponds to a GSD graph in the above sense (i.e. for the transformed Π', where H_2 plays the role of the random oracle) , with the only difference that there are additional edges corresponding to a second random oracle H_1. The following Theorem shows that this difference does not impact security.

Theorem 18. *If the encryption scheme in TTKEM is $(t, \tilde{\epsilon})$-IND-CPA secure and H_1, H_2 are modelled as random oracles, then TTKEM is (Q, M, t, ϵ)-CGKA-secure, where $\epsilon = \tilde{\epsilon} \cdot 8(MQ)^2 + \mathsf{negl}(\lambda)$ (where λ denotes the seed length), where M is an upper bound on the group size.*

Sketch. In order to adapt the proof of Theorem 17 to the setting where some seeds may be derived from others by the random oracle H_1, we first slightly change the event E to also include queries to H_1, i.e. the event E is now that A queries H_1 *or* H_2 on a seed that it doesn't trivially know through corruptions or the challenge query. The case distinction in the main proof on the probability of A remains the same.

The case where E happens in GSD_0 with relatively small probability $< \epsilon/2$ (cf. Lemma 23) is easily handled, since in this case the reduction may generate all seeds independently as before and when nodes get corrupted, it can simply program H_1 to ensure consistency. The seeds in the challenge graph are not consistent in this simulation, but since the adversary does not query any of these seeds with high (enough) probability, this is indistinguishable and our reduction retains most of the advantage.

The other case, where E happens in GSD_0 with large probability $\geq \epsilon/2$ is handled similarly. The key observation in Lemma 24 is that it is sufficient to simulate the GSD game correctly until E happens. Using the approach of programming H_1, this is still the case and thus Corollary 4 also holds in this case.

We conclude by observing that the CGKA graph has at most $N = 2MQ$ nodes and that none of the seeds and secret keys in the challenge graph (for any safe group key) is leaked by Lemma 21. □

We remark that, similarly to the previous proof, one can easily adapt it to the case of TreeKEM (with blanking) since the loss in security only depends on the maximal number of nodes in the challenge graph but not on its structure.

5.4 Open Problems

In this chapter we provided the first adaptive security reductions with subexponential loss for any CGKA protocol; all previous adaptive security proofs for various CGKA constructions suffered from an exponential loss in security. More precisely, for TTKEM/TreeKEM we presented an adaptive security proof with only a quasipolynomial loss in the standard model. It remains an exciting open problem whether this upper bound could be strenghened to a polynomial. While we do achieve a small polynomial bound in the random oracle model, it seems hard to improve on our bounds in the standard model. In fact, in Chapter 7 we exploit inherent limitations to currently used proof techniques and prove that no non-rewinding blackbox reduction can prove adaptive security of TreeKEM/TTKEM based on the IND-CPA security of the underlying encryption scheme with only a polynomial loss. Hence, in order to achieve adaptive security for CGKA with only polynomial loss in the standard model, potential directions could be to either 1) use more sophisticated (rewinding/non-blackbox) reductions, 2) make stronger assumptions on the underlying encryption scheme, or 3) come up with completely new constructions of CGKA.

CHAPTER 6

Adaptive Indistinguishability of Yao's Garbling

6.1 Introduction

Suppose that Alice, who holds a function represented as a Boolean circuit C, and Bob, who holds an input x to that function, want to jointly evaluate $y = \mathsf{C}(x)$ such that Alice learns nothing about x while Bob learns nothing about C (except for some side-information that is unavoidable). Yao put forward[1] the following elegant solution:

1. Alice first sends $\widetilde{\mathsf{C}}$, a "garbling" of the circuit C, to Bob,
2. Bob then obtains $\widetilde{x}$, a "garbling" of his input x, from Alice via oblivious transfer,
3. Bob finally evaluates $\widetilde{\mathsf{C}}$ on $\widetilde{x}$ to learn y and sends it over to Alice.

Yao showed how the garbling steps above can be carried out using a symmetric-key encryption (SKE) scheme – and hence based on the minimal assumption of one-way functions. This has been referred to as Yao's garbling scheme, and is the focus of this work. We describe it next in slightly more details.

Yao's garbling scheme. Let $(\mathsf{Enc}, \mathsf{Dec})$ be a (special) SKE. To garble a circuit $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$ with fan-in 2 and arbitrary fan-out:

1. Alice first samples a pair of secret keys $(\mathrm{k}_w^0, \mathrm{k}_w^1)$ for each wire w in C.
2. For every gate $g : \{0,1\}^2 \to \{0,1\}$ with left input wire u, right input wire v, and output wire w, she then computes a *garbling table* $\widetilde{g}$ consisting of the four ciphertexts listed in Table 6.1.(a), in randomly permuted order.

This Chapter essentially replicates, with permission, the full version [KKP21b] of our publication [KKP21a], © IACR 2021, to appear.

[1] According to [BHR12b], the idea was first presented by Yao in oral presentations on secure function-evaluation [Yao82, Yao86] but formally described only in [GMW87].

$\mathsf{Enc}_{\mathrm{k}_u^0}(\mathsf{Enc}_{\mathrm{k}_v^0}(\mathrm{k}_w^{g(0,0)}))$ $\mathsf{Enc}_{\mathrm{k}_u^1}(\mathsf{Enc}_{\mathrm{k}_v^0}(\mathrm{k}_w^{g(1,0)}))$ $\mathsf{Enc}_{\mathrm{k}_u^0}(\mathsf{Enc}_{\mathrm{k}_v^1}(\mathrm{k}_w^{g(0,1)}))$ $\mathsf{Enc}_{\mathrm{k}_u^1}(\mathsf{Enc}_{\mathrm{k}_v^1}(\mathrm{k}_w^{g(1,1)}))$	$\mathsf{Enc}_{\mathrm{k}_u^0}(\mathsf{Enc}_{\mathrm{k}_v^0}(\mathrm{k}_w^{0}))$ $\mathsf{Enc}_{\mathrm{k}_u^0}(\mathsf{Enc}_{\mathrm{k}_v^1}(\mathrm{k}_w^{0}))$ $\mathsf{Enc}_{\mathrm{k}_u^1}(\mathsf{Enc}_{\mathrm{k}_v^0}(\mathrm{k}_w^{0}))$ $\mathsf{Enc}_{\mathrm{k}_u^1}(\mathsf{Enc}_{\mathrm{k}_v^1}(\mathrm{k}_w^{0}))$	$\mathsf{Enc}_{\mathrm{k}_u^0}(\mathsf{Enc}_{\mathrm{k}_v^0}(\mathrm{k}_w^{1}))$ $\mathsf{Enc}_{\mathrm{k}_u^0}(\mathsf{Enc}_{\mathrm{k}_v^1}(\mathrm{k}_w^{1}))$ $\mathsf{Enc}_{\mathrm{k}_u^1}(\mathsf{Enc}_{\mathrm{k}_v^0}(\mathrm{k}_w^{1}))$ $\mathsf{Enc}_{\mathrm{k}_u^1}(\mathsf{Enc}_{\mathrm{k}_v^1}(\mathrm{k}_w^{1}))$
(a)	(b)	(c)

Table 6.1: Garbling tables for (a) general gate g (b) constant-0 gate and (c) constant-1 gate. u and v denote the two input wires and w denotes the output wire. The two keys associated with (say) the wire u are denoted by k_u^0 and k_u^1.

3. Finally, she constructs the *output mapping* μ which, for each output wire w, maps each of the keys $(\mathrm{k}_w^0, \mathrm{k}_w^1)$ to the bit it "encodes".

The *garbled circuit* $\widetilde{\mathsf{C}}$, which Alice sends it over to Bob, consists of all the garbling tables $\widetilde{g}$ and the output map μ. This constitutes the *offline* phase of the protocol. To garble an input $x = x_1 \| \ldots \| x_n$, Alice simply gives out, for each input wire w_i, the key $\mathrm{k}_{w_i}^{x_i}$ corresponding to the bit x_i. This constitutes the *online* phase of the protocol. To evaluate the garbled circuit on the garbled input, the encryption scheme must satisfy a *special correcntess property*: for each ciphertext $\mathrm{c} \leftarrow \mathsf{Enc}_{\mathrm{k}}(\mathrm{m})$ there should exist a single key (i.e., k) such that decryption passes. Using the keys in the garbling input, Bob can now evaluate C "over the encryption" as follows:

1. Starting from the input level and in some topological order, he progressively decrypts each garbling table in $\widetilde{\mathsf{C}}$ by trying the two keys in hand on all the four ciphertexts for each garbling table. Thus, in each step, he learns one of the secret keys corresponding to the output wire of the gate in consideration.

2. At the end of this process, Bob recovers exactly one of the two keys associated with each output wire of the circuit. This allows him to use the output map μ to "decode" the revealed output keys to the output string $y \in \{0,1\}^m$.

The scheme as described above is what is regarded to be the original formulation of Yao's garbling scheme [LP09, JW16]. A slight variant in which Alice defers sending the output map μ to the online phase (along with $\widetilde{x}$) is also of interest [JW16], although it suffers from a higher online complexity compared to the original formulation. To avoid confusion, we refer to the original scheme as *Yao's offline garbling scheme* and the modified scheme as *Yao's online garbling scheme* or, in short, online Yao and offline Yao respectively. Our work concerns the security of offline Yao.

Security. Even though garbling schemes found several applications (see [BHR12b]), its security was formally analyzed much later in [LP09]. They consider a simulation-based notion[2] captured by the following experiment:

1. The adversary submits a circuit-input pair (C, x) to the challenger.

[2]This is an equivalent formulation of the definition in [LP09] and is taken from [JW16]. Our overview of the proof in [LP09] to be discussed in Section 6.1.2 has been adapted accordingly.

2. The challenger responds either with the real garbling $(\widetilde{\mathsf{C}}, \widetilde{x})$ (i.e., real game or Real) or with a "simulated" garbling where a constant-0 circuit is used instead of C (i.e., simulated game or Sim). The constant-0 circuit has the same topology as C but with all its gates replaced by constant-0 gates.

3. The adversary wins if it guesses which case it is.

Then they gave a reduction from the (special) indistinguishability of the underlying SKE for *offline* Yao. Note that the adversary in the above security game must *select* the garbling input x at the same time as the circuit C. This is in conflict with the online-offline nature of the actual scheme where Bob (a potential adversary) sees $\widetilde{\mathsf{C}}$ *before* he commits to x. Hence Bob could have chosen the input *adaptively*, based on $\widetilde{\mathsf{C}}$. In fact, such a scenario does arise in applications such as one-time programs and secure outsourcing [BHR12a]. Therefore it is natural to consider strengthening the above selective definition of simulatability to an adaptive definition where A gets to choose the input after it sees the garbling of a circuit of its choice. Unfortunately, as it was shown in [AIKW13] this is too strong a notion to attain for offline Yao. This is a consequence of their more general negative result that the *online complexity* of a garbling scheme (or, more generally, a randomised encoding scheme) in the adaptive setting *must* exceed the output-size of the circuit (given that one-way functions exist).[3] Jafargholi and Wichs [JW16] observed that this negative result does not apply to online Yao since the output map there gets sent in the online phase, and even managed to prove adaptive *simulatability* of online Yao. While security of other variants of Yao's garbling scheme was also proven [HJO$^+$16, JSW17], the case of offline Yao was largely ignored.

6.1.1 Our Results

Although the negative result in [AIKW13] rules out adaptive simulatability of offline Yao, it is not clear if it also applies to its *adaptive indistinguishability* [BHR12b], which is defined by the following experiment:

1. The adversary submits a pair of circuits $(\mathsf{C}_0, \mathsf{C}_1)$ of the same topology to the challenger.

2. The challenger flips a coin b and responds with $\widetilde{\mathsf{C}}_b$.

3. The adversary then submits a pair of inputs (x_0, x_1) such that $\mathsf{C}_0(x_0) = \mathsf{C}_1(x_1)$ and the challenger responds with $\widetilde{x}_b$.

4. The adversary wins if it guesses the bit b correctly.

Although it is a weaker notion of security, adaptive indistinguishability suffices for certain applications (e.g., adaptively-indistinguishable symmetric-key functional encryption [JSW17]).

[3] On a high level, their argument for Offline Yao proceeds as follows. Consider garbling a PRG G (of sufficient stretch) and suppose that $|\widetilde{x}|$ is less than that of output of the PRG. Then the (online) simulator for such a scheme can be used to break the security of G. This exploits the fact that the $\widetilde{x}$ loses information when it encodes an element in the co-domain of G that is not an image (but not for an image). It is possible to circumvent this result in the random-oracle model [BHR12a].

	Selective		Adaptive	
	Offline Yao	Online Yao	Offline Yao	Online Yao
Simulatability	[LP09]		[AIKW13]	[JW16]
Indistinguishability			This work	

Table 6.2: Security of the two variants of Yao's garbling. The (only) negative result is highlighted in red.

Our results. We help (partially) complete the landscape for security of Yao's garbling (see Table 7.1). To this end, we characterise the adaptive indistinguishability of offline Yao in terms of the treewidth[4] of the circuit. Our main results are informally stated below.

Theorem (main)**.** *Consider the class of Boolean circuits $\mathcal{C}$ of size N with treewidth $w = w(N)$. Offline Yao is adaptively indistinguishable for $\mathcal{C}$ with $N^{O(w)}$ loss in security.*[5]

For Boolean circuits of constant (resp., poly-logarithmic) treewidth, we obtain the following corollary.

Corollary. *Offline Yao is adaptively indistinguishable for Boolean circuits of size N and $O(1)$ (resp., $\mathrm{polylog}(N)$) treewidth with a polynomial (resp., quasi-polynomial) in N loss in security.*

Finally, exploiting a classical result from graph theory – that planar circuits of size N have treewidth at most $\sqrt{N}$ [LT79] – we obtain the following corollary.

Corollary. *Offline Yao is adaptively indistinguishable for planar Boolean circuits with a sub-exponential loss in security.*

Interpreting our results. Treewidth is a notion from algorithmic graph theory that has found several applications in parametrised and circuit complexity (see Section 6.1.3) Intuitively, it is a (graph) property that measures how "far" the circuit is from a formula (and, more generally, how far a graph is from a tree): in particular, the smaller the treewidth the closer the circuit is to a formula. Therefore, it is not surprising that having a low treewidth limits how powerful a circuit can be. A precise characterisation of this (from above) was given in [GJ16]: every circuit of size N and treewidth $w = w(N)$ can be simulated in *depth* $w \log(N)$. Thus, e.g., circuits of constant treewidth can be simulated in NC^1. Whether the converse is true in general – i.e., whether NC^i can be simulated using circuits with treewidth $O(\log^{i-1}(N))$ – is an open problem to the best of our knowledge.[6] However it is partially true: namely, NC^1 circuits can be simulated using polynomial-sized Boolean formulae (which, by definition, have treewidth 1) [Spi71, Bre74]. Consequently, the first corollary applies to functions computable in NC^1.

Given the aforementioned negative result from [AIKW13], we find any proof of adaptive security for offline Yao rather surprising. Nevertheless, there are scenarios where our results also lead to improvements in concrete efficiency (even after the loss in security is taken into account).

[4] Since treewidth is defined for undirected graphs, whenever we refer to the treewidth of a directed graph (or a circuit) we refer to the treewidth of the graph obtained by ignoring the direction of its edges.

[5] Consider a reduction R from a problem P to another problem Q. Suppose that R uses an (t, ϵ)-adversary A that breaks Q in order to (ϵ', T')-break P. Informally, the *loss in security* incurred by R can be defined as the ratio $(\epsilon T')/(\epsilon' T)$.

[6] See this question (48504) posted on CSTheory, Stack Exchange.

We describe one such scenario next. Recall from the discussion above that for functions computable in NC^1, we show security of offline Yao at only a polynomial loss. Moreover, the online complexity of garbling such a function using offline Yao depends only on the input length n (times the security parameter λ). Now, note that PRGs of arbitrary stretch (say n^c for a constant $c \in \mathbb{N}$) exist in NC^1 [CM01, IN96]. However, if one were to use online Yao, then the online complexity is substantial ($n^c \times \lambda$). This example is particularly interesting since offline Yao for such a function is not simulatable at all as a consequence of the negative result.

Implications to simulatability of online Yao. It is worth pointing out that our results may also imply tighter reductions for simulatability of online Yao. The reduction for simulatability of online Yao from [JW16] loses a factor that is exponential in the *width* of a circuit: our approach can be seen as an extension of their techniques. Since treewidth is bounded from above by width, in cases where there is a gap between treewidth and width for a circuit class, our approach would lead to a tighter reduction for simulatability of online Yao compared to [JW16]. A more detailed explanation follows later in Remark 3.

Comparison with [JSW17]. We conclude the section by comparing our result with [JSW17], which is also concerned with adaptively-indistinguishable garbled circuits. The construction in [JSW17] builds on [HJO$^+$16] and therefore has offline Yao as its basis. However, it requires (i) applying an additional layer of *somewhere equivocal encryption* to the garbling table and (ii) modifying the circuit to be garbled in order to make the security proof go through. These modifications lead to their construction being less efficient compared to plain offline Yao, but it does allow them to prove adaptive indistinguishability. It is not clear if any of the ideas employed there can be used to argue the indistinguishability of offline Yao (this is, in fact, posed as an open question there).

6.1.2 Technical Overview

Outline. Our starting point is the reduction proving adaptive simulatability of online Yao [JW16]. The key idea in [JW16] is to abstract out the hybrid argument using a pebbling game on the circuit, which we call the black-grey (BG) pebbling game (Definition 46). To be precise, they showed that if a circuit allows a BG pebbling strategy of length τ that uses σ (black) pebbles, then there exists a reduction proving adaptive simulatability of online Yao with a loss in security at most $O(\tau 2^\sigma)$. This allows us to shift the focus from security reductions to the conceptually-cleaner task of coming up with "pebble-efficient" strategies. We start off below by describing this connection and then explain why this approach falls short when it comes to arguing adaptive indistinguishability (or simulatability) of offline Yao. Next we show how this issue can be remedied, key to it is a new pebbling game, which we call the black-grey-red (BGR) pebbling game (Definition 50). Analogous to [JW16], we prove that if there exists a BGR pebbling strategy of length τ that uses σ ("greyscale", i.e. black or grey) pebbles, then there exists a reduction proving adaptive indistinguishability of offline Yao where the loss in security is at most $O(\tau 2^\sigma)$ (Theorem 24). Finally to complete the proof – and as our main technical contribution – we describe a pebble-efficient strategy for the BGR pebbling game in which the number of (greyscale) pebbles used grows only with the treewidth of the circuit (Theorem 23). The strategy has a divide-and-conquer flavour and crucially relies on the notion of *separators* from graph theory. We next elaborate on each of the steps above.

Pebbling game and hybrids

The reduction in [JW16] builds on the reduction for selective simulatability of offline Yao [LP09]. Both these works follow a sophisticated hybrid argument which can be described abstractly using a BG pebbling strategy.

Pebbles and garbling modes. The BG pebbling game, as its name suggests, uses two types of pebbles: black and grey. A pebbling configuration $\mathcal{P}$ for a circuit C determines how the garbled circuit $\widetilde{\mathsf{C}}$ is simulated in the hybrid $\mathsf{H}_{\mathcal{P}}$. To be more precise, the pebbling configuration $\mathcal{P}$ can associate each gate g in C with a black or grey pebble. In order to translate $\mathcal{P}$ to the garbling $\widetilde{\mathsf{C}}$, the simulator in hybrid $\mathsf{H}_{\mathcal{P}}$ does the following:

- if g carries no pebble in $\mathcal{P}$, then the corresponding garbling table in $\widetilde{\mathsf{C}}$ consists of an honest garbling table of g (Table 6.1.(a))
- if g carries a grey pebble, then the garbling table encodes a constant-0 gate (Table 6.1.(b)).
- if g carries a black pebble then the garbling table encodes either a constant-0 or a constant-1 gate (Table 6.1.(c)) *depending* on the value of (the output wire of) g when C is run on the garbling input x.

The three *modes* above of simulating individual gates are named real, simulated and input-dependant modes respectively or, for short, Real, Sim and Input, respectively (Table 6.3.(a)). Note that the real garbling game corresponds to the empty pebbling configuration (since all the gates are honestly garbled), whereas the simulated game will correspond to the all-grey configuration (since all the gates have been replaced by the constant-0 gate).

Pebbling rules. Note that any arbitrary configuration of pebbles $\mathcal{P}$ describes a valid hybrid $\mathsf{H}_{\mathcal{P}}$. The role of the pebbling rules is to model indistinguishability of neighbouring hybrids. To be more precise, if a pebbling configuration $\mathcal{Q}$ can be obtained from another configuration $\mathcal{P}$ by a valid pebbling move (or vice versa) then the hybrids $\mathsf{H}_{\mathcal{P}}$ and $\mathsf{H}_{\mathcal{Q}}$ should be indistinguishable. Consequently a BG pebbling strategy $\boldsymbol{\mathcal{P}}$, which must start from an empty configuration and end with the all-grey configuration, leads to a valid sequence of hybrids that establishes that the real garbling game and simulated garbling game are indistinguishable, proving the security of the garbling scheme. In the BG pebbling game, the following moves (see Figure 6.1) are allowed:

1. a black pebble can be placed on or removed from a gate g if and only if g's predecessor gates are pebbled black; and
2. a black pebble on a gate g can be replaced by a grey pebble if g's *successor* gates are pebbled, either black or grey.

To understand the rationale behind the two rules, one needs to take a closer look at the structure of a garbling table in Yao's scheme. Since this is not that relevant to the current discussion, we refer the readers interested in more details to Section 6.3.

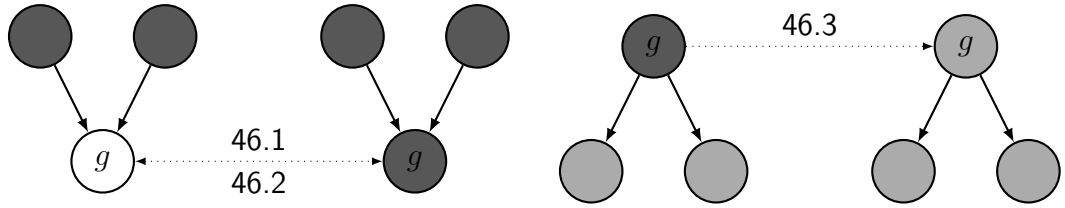


Figure 6.1: Rules for the BG pebbling game.

Selective Simulatability of Offline Yao. Observe that in order to simulate $\widetilde{\mathsf{C}}$ in a hybrid $\mathsf{H}_{\mathcal{P}}$, the simulator only needs to know the output value of those gates that are pebbled black in $\mathcal{P}$ (i.e., the gates in Input mode). In the selective setting, since the adversary commits to the garbling input x in the offline phase, the value of *all* the gates is available beforehand. Hence, in this case the simulator has the luxury of using as many pebble as it needs. Therefore the pebbling strategy (implicitly) employed in [LP09] is the following:

1. starting from the input gates, pebble the circuit completely black in some topological order, and then
2. starting from the output gates and in reverse topological order, replace each black pebble with a grey pebble.

To complete the description of the hybrid $\mathsf{H}_{\mathcal{P}}$ in the selective setting, one thing remains to be addressed: For concreteness, let's consider the simulated game, which corresponds to the all-grey pebbling configuration (the argument for other hybrids is analogous). Note that it is not possible to send the honestly-generated output map μ in $\mathsf{H}_{\mathcal{P}}$ since this will lead to the output being mapped to the all-0 string. However, since x is available in the offline phase, [LP09] resolved this issue by *programming* the output map to map the zero-keys of the output wires to $\mathsf{C}(x)$. The adversary cannot tell this from the honest output map since the change is information-theoretic.

Since the above pebbling strategy takes at most $2N$ moves (and uses N black pebbles) the corresponding hybrid argument only loses a $2N$ factor. It is possible to further reduce to adaptive simulatability via *random guessing*, but this incurs an additional loss in security that is exponential in the length of x.

Adaptive Simulatability of Online Yao. In order to avoid this exponential loss in the adaptive setting, [JW16] had to mainly tackle two issues, both arising from the fact that the garbling input x is now only available in the online phase.

1. Firstly, simulating the hybrids could not rely on the knowledge of the values of too many gates in C.
2. Secondly, the output map could no longer be programmed in the offline phase since the output $\mathsf{C}(x)$ is only determined in the online phase.

The first issue was resolved in [JW16] by employing BG pebbling strategies that were more frugal in terms of the number of black pebbles used. To this end, they proved that if there exists a BG pebbling strategy of length τ that uses σ black pebbles, then the loss in the resulting security is at most $O(\tau 2^{\sigma})$. Here, loosely speaking, the 2^{σ} factor is the cost of

randomly guessing the output values of the gates pebbled black, which they require in order to carry out the simulation of the hybrids (as well as the reduction).[7] To complete their proof, [JW16] described two (generic) pebbling strategies: one where σ grows only with the width of the circuit and another where σ grows only with the depth of the circuit. A consequence of the latter is the adaptive simulatability of log-depth (i.e., NC^1) circuits with a *polynomial* loss in security.

The second issue, on the other hand, was basically side-stepped by *modifying* the garbling scheme to defer the sending of the output map to the online phase, i.e., by resorting to *online* Yao. This tweak allowed [JW16] to carry out a "deferred programming" of the output map since the garbling input is available in the online phase. The cost is an increased online complexity which is now dependent also on the output size.

Indistinguishability of Offline Yao: Our Approach.

Unfortunately, given the negative result from [AIKW13], it is unlikely that a result as strong as [JW16] could be shown for adaptive simulatability of offline Yao.[8] However, as we will see, relaxing the security requirement to adaptive indistinguishability offers some wiggle room. The key to exploiting this, as we explain next, is to discard the simulated garbling mode (Sim) in the hybrids altogether, which allows us to argue security *without* having to program the output map.

Bypassing the simulated mode. A standard way to show that a simulation-based definition implies an indistinguishability-based definition (e.g., think of semantic security and IND-CPA) is to use a two-step hybrid argument where the simulated game acts as an intermediary between the "left" and "right" indistinguishability games. If one attempts to use this approach in our context and use the result from [JW16] to argue adaptive indistinguishability of *offline* Yao garbling, we immediately run into the issue with programming the output map. Thus it seems that the necessity to program the output map is tied to the simulated game, and hence to the simulated mode of garbling. The main idea behind our reduction is therefore to avoid the simulated mode and instead only work with the real and input-dependant modes, which do not require programming the output map. Thus in all our hybrids, the output map is simply the honestly-generated output map and therefore can be generated in the offline phase itself.

Our approach. Our idea is to directly replace – gate by gate – the honest garbling table of gates in C_0 (Real_0) with that of gates in C_1 (Real_1). Since the luxury of programming the output map is no longer available, it is crucial to ensure that the evaluation of the garbled circuit in all intermediate hybrids is correct at all times: even though $\mathsf{C}_0(x_0) = \mathsf{C}_1(x_1)$ holds (by definition) there is no guarantee that the output of the internal gates of C_0 and C_1 match. An error propagated as a result of one circuit influencing the computation of another may render the hybrids trivially distinguishable to the adversary (via evaluation of the garbling). To this end, we employ the input-dependant modes for (C_0, x_0) and (C_1, x_1) (resp., Input_0 and Input_1). In more details, in all our hybrids, we ensure that a gate in Real_0 mode is *never* adjacent to another gate in the Real_1 mode. This is accomplished by maintaining a "frontier" of gates in Input_0 and Input_1 mode in between the gates in real mode (see Figure 6.5). This

[7] This is one of the earliest applications of the Piecewise-Guessing framework from Chapter 3.

[8] Since pseudo-random generators (of arbitrary stretch) exist in NC^1 [CM01, IN96], the result in [AIKW13] rules out reductions with polynomial loss for *offline* Yao. This is in stark contrast to the aforementioned positive result from [JW16] for *online* Yao for NC^1 circuits.

separation of the left (Real_0 and Input_0) and right (Real_1 and Input_1) modes guarantees that the computations belonging to the two circuits do not "corrupt" each other. We point out that this is reminiscent of (circuit) simulation strategies adopted in certain works in circuit complexity [GJ16] (see Section 6.1.3).

The design of our black-grey-red (BGR) pebbling game is carried out keeping the above blueprint in mind. Looking ahead, one can think of it as a symmetrised formulation of the BG pebbling game. Our proof that a BGR strategy implies a valid sequence of hybrids is mostly similar to that in [JW16]: we show that if there exists a BGR pebbling strategy of length τ that uses σ greyscale pebbles, then there exists a reduction to adaptive indistinguishability of offline Yao with a loss in security at most $O(\tau 2^{\sigma})$ (Theorem 24).[9] The bulk of our technical work goes into coming up with pebble-efficient strategies for the BGR pebbling game. This task turns out to be considerably more involved than for the BG pebbling game (primarily due to the constraints introduced by the additional rules in the BGR game). The best strategy we could come up with exploits the treewidth w of the circuit, and as a result the number of (greyscale) pebbles used is roughly $\sigma := w\delta \log(N)$, where N is the size of the circuit and δ its fan-out. The strategy has a divide-and-conquer flavour and crucially relies on the notion of separators from graph theory [RS86, Bod98]. In the remainder of the technical overview, we informally present the BGR pebbling game and then briefly explain the treewidth-based BGR strategy.

BGR pebbling game. Let g denote the location of a gate in $G := \Phi(\mathsf{C}_0) = \Phi(\mathsf{C}_1)$, the directed acyclic graph (DAG) underlying the circuits, and let g_0 (resp., g_1) denote the corresponding gate in C_0 (resp., C_1). The BGR pebbling game, as its name suggests, uses three types of pebbles: black, grey and red. In order to translate a BGR pebbling configuration $\mathcal{P}$ to the garbling $\widetilde{\mathsf{C}}$, the simulator in hybrid $\mathsf{H}_{\mathcal{P}}$ does the following for all internal gates g:

- if g carries no pebble in $\mathcal{P}$, then its garbling table in $\widetilde{\mathsf{C}}$ will be the honest garbling table of g_0,
- if g carries a black pebble then the honest garbling table will be replaced by that of constant-0 or constant-1 gate *depending* on the output value of g_0 when C_0 is run on x_0,
- if g carries a grey pebble, then the simulation is the same as in previous case except that the garbling *depends* on the output value of g_1 when C_1 is run on x_1,
- if g carries a red pebble, then its garbling table in $\widetilde{\mathsf{C}}$ will be the honest garbling table of g_1.

The input is then garbled as follows: For the ith input gate, if this gate carries no pebble or a black pebble, then the ith key in $\widetilde{x}$ is the key corresponding to the ith bit of x_0, otherwise it is the key corresponding to the ith bit of x_1. (The pebbles on the output gates are simply ignored.) The four modes of simulation above are real and input-dependant modes for the left and right game respectively or, in short, Real_0, Input_0, Input_1 and Real_1 respectively (see Table 6.3.(b)). Note that the semantics of gates that carry no pebble or a black pebble is the same as in the BG pebbling game (if one sets $(\mathsf{C}_0, x_0) = (\mathsf{C}, x)$), but a grey pebble is now interpreted differently. A BGR pebbling strategy starts off with a configuration with all

[9] We use the Piecewise-Guessing framework from Chapter 3 instead of a direct argument as in [JW16].

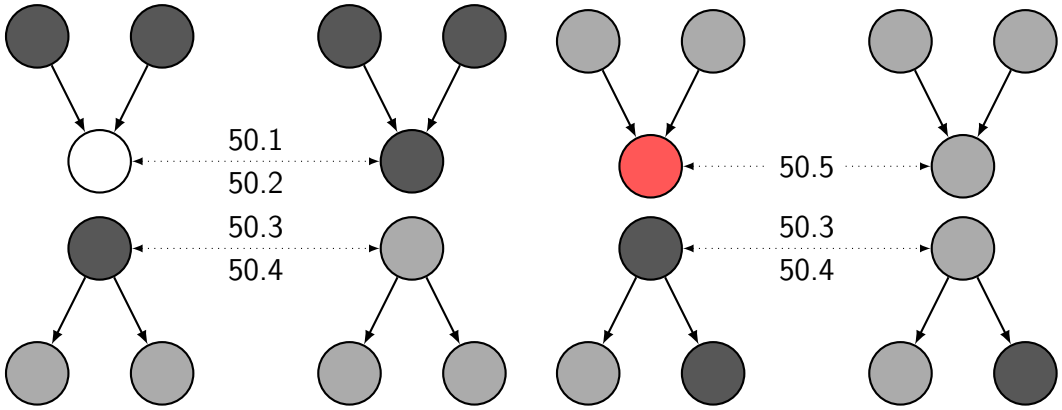


Figure 6.2: Rules for the BGR pebbling game.

gates empty (i.e., honest garbling of C_0) but the goal is now to pebble them all red (i.e., honest garbling of C_1). Thus the extreme hybrids correspond to the left and right games in the adaptive indistinguishability game. The pebbling rules, listed below (see Figure 6.2), are designed keeping the above discussion in mind and so that indistinguishability of neighbouring hybrids can be argued (Lemma 25):

1. a black pebble can be placed on or removed from a gate g if and only if g's predecessor gates are pebbled black; and
2. a black pebble on a gate g can be *swapped* with a grey pebble if g's successor gates are pebbled, either black or grey; and
3. a grey pebble on a gate g can be *swapped* with a red pebble if g's predecessor gates are pebbled grey.

Note that the dynamic between no pebbles and black pebbles is similar to the dynamic between red and grey pebbles (hence the reason we consider it to be a symmetric version of the BG pebbling game). Since the outpt values of gates which carry a black or grey pebble in $\mathcal{P}$ need to be known to carry out the simulation of $\mathsf{H}_{\mathcal{P}}$, the goal here is to minimise the number of such "greyscale" pebbles.

Treewidth, separators and BGR pebbling strategies. Compared to the BG pebbling game, pebble-efficient strategies for the BGR pebbling game are harder to come by. (This is not surprising, in hindsight, given the negative result [AIKW13].) In particular, the generic pebbling strategies used in [JW16] no longer work without incurring a blow-up in the number of pebbles employed.[10] Below we briefly explain our treewidth-based strategy, the best (generic) strategy we could come up with.

Crucial to our strategy is the notion of *separators*. Informally, a separator for a circuit C of size N is a subset of gates $\mathcal{S}$ such that removing $\mathcal{S}$ (and the edges incident on it) from C partitions C into sub-circuits of "comparable" size. Slightly more formally, $\mathcal{S}$ partitions C into sub-circuits $\mathsf{C}_1, \ldots, \mathsf{C}_p$ such that for every sub-circuit C_i, $|\mathsf{C}_i| \leq 2/3 \cdot N$ (say). In a classical result from graph theory, it is shown that the size of separator of a graph (and

[10] The width-based BG strategy from [JW16, HJO+16] can be modified to obtain a comparable BGR strategy for *levelled* circuits. However, the resulting security bounds do not yield any advantage over simply guessing the input (which we want to avoid).

therefore a circuit) is at most its treewidth [RS86, Bod98]. Since treewidth is a monotonous property – i.e., removing wires or gates from C can only decrease its treewidth – the process of decomposition into sub-circuits using separators can be recursively carried out further (using a different separator each time) till one ends up with constant-size sub-circuits. Such a recursive decomposition is also carried out in the simulation in [GJ16, Theorem 2] (also see [Bod88]).

Our pebbling strategy exploits this recursive decomposition to minimise the number of greyscale pebbles used. To this end, the pebbling strategy maintains *long-term* greyscale pebbles only at the separators. These pebbles help reduce the task at hand to that of (recursively) pebbling the resulting sub-circuits, one at a time *reusing* pebbles in that process. Therefore, our pebbling strategy can be recursively described as follows:

- place greyscale pebbles at the separator $\mathcal{S}$ of G,
- recursively, one at a time, place red pebbles on each subcircuit C_i,
- replace the greyscale pebbles on $\mathcal{S}$ with red pebbles.

Since the depth of the recursion is bounded by $O(\log N)$ (thanks to the property of the separator), the hope would be that the number of greyscale pebbles maintained overall does not blow up. We show that this is indeed the case as our main technical contribution (Theorem 23).

Theorem (main)**.** *Any circuit* C *of size* N*, fan-out* δ *and treewidth* w *can be BGR pebbled using* $O(\delta w \log(N))$ *greyscale pebbles.*

Translating the above divide-and-conquer approach into an actual pebbling strategy (Section 6.4) turns out to be tricky due to the intricate nature of the BGR pebbling rules. Therefore, before presenting the strategy for arbitrary graphs (Section 6.4.2), we present a BGR pebbling strategy for path graphs (P) as a warm-up (Section 6.4.1) – since paths have treewidth 1, the latter can be viewed as a special case of the former. We refer the readers to Section 6.4 for the details.

Epilogue. It is instructive to review the above pebbling strategy in terms of the actual simulation. The (garbling tables of) circuit C_0 is being progressively, piece by piece, replaced by the (garbling tables of) circuit C_1 as dictated by the recursion, with the bulk of the replacement happening at the base of the recursion. It is exactly those long-term greyscale pebbles placed on the separators which act as the frontier between the pieces of C_0 and C_1. This ensures that computations of the two circuits are insulated from each other (see Figure 6.5).

Remark 3. We remark that our result on the BGR pebbling complexity can also be used to prove tighter security bounds for simulatability of online Yao for circuit classes where the treewidth is smaller as the width. This is true since any BGR sequence with complexity σ implies a BG sequence with complexity at most σ: simply consider the BG sequence obtained from a BGR sequence by substituting all the red pebbles with a grey pebble, and note that for BG pebbling only the number of black pebbles is counted.

6.1.3 Further Related Work

Adaptive security for garbled circuits. The problem of constructing adaptively-secure garbling schemes was first raised by Bellare, Hoang and Rogaway in [BHR12a]; they gave a first adaptively-secure construction in the random oracle model, which bypasses the lower bound of Applebaum et al. [AIKW13]. Bellare, Hoang and Keelveedhi [BHK13] then proved the previous scheme adaptively-secure in the standard model, but under non-standard assumptions on hash functions. Further constructions from various assumption followed: Boneh et al. [BGG$^+$14] constructed an adaptively-secure scheme from the learning with errors (LWE) assumption, where the online complexity depends on the depth of the circuit family. Ananth and Sahai [AS16] constructed an optimal garbling scheme from iO. Later, Ananth and Lombardi [AL18] constructed succinct garbling schemes from functional encryption. In [JSW17], Jafargholi et al. relax the simulation-based security to *indistinguishability* and show how to construct adaptively-secure garbling schemes from the minimal assumption of one-way functions, where the online complexity only depends on the pebble complexity and the input-size, but is *independent* of the output-size. A particularly strong result in this area was due to Garg and Srinivasan [GS18], who constructed adaptively-secure garbling with near optimal online complexity that can be based on standard assumptions such as the computational Diffie-Hellman (CDH), the factoring, or the LWE assumption. While this list is far from complete, we finally mention a recent work by Jafargholi and Oechsner [JO20] who analyze adaptive security of several practical garbling schemes. They give positive as well as negative results, and argue why the techniques from [JW16] cannot be applied to certain garbling schemes.

Treewidth, Separators and Computational Complexity. Treewidth [RS86] has its roots in algorithmic graph theory. Many hard graph-theoretical problems become tractable when one restricts to graphs of bounded treewidth. In some cases, this leads to even NC algorithms for problems which are otherwise known to be NP-complete (e.g., [Bod88]). More often than not, this is because bounding the treewidth leads to divide-and-conquer algorithms, sometimes via separators (see [Bod93] for an instructive survey). Unsurprisingly, this also has several consequences in circuit complexity (e.g., [AR11, ACL$^+$14, LMPP18]), and perhaps the most relevant to our work are [JS14, GJ16]. It was shown in [JS14] that circuits with constant treewidth can be simulated in NC^1; [GJ16] extended this result by using separators to show that circuits of size N and treewidth w can be simulated in depth $w \log(N)$. Both these results can be regarded as a generalization of Spira's theorem that Boolean formulae can be simulated by NC^1 circuits [Spi71].

6.2 Preliminaries

6.2.1 Notation for circuits

We consider Boolean circuits with explicit input and output gates, associated with the input and output wires respectively. For a circuit $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$ with N gates (including the n input and m output gates) and W wires of which n (resp., m) are input (resp., output) wires, we denote the DAG that represents the topology of the circuit C by $\Phi(\mathsf{C})$. That is, $\Phi(\mathsf{C})$ is a graph with $\mathcal{V} = [1, N]$ obtained by:

1. assigning the input (resp., output) gates to the vertices $[1, n]$ (resp., $[N - m + 1, N]$),
2. assigning the internal gates to the vertices $[n + 1, N - m]$, and

3. assigning the wires of the circuit to the edges.

The wires are assigned an index from $[1, W]$, with the input (resp., output) wires indexed from $[1, n]$ (resp., $[W - m + 1, W]$). An internal gate of a circuit is represented by a four-tuple (g, u, v, w) where $g : \{0,1\}^2 \to \{0,1\}$ denotes the predicate implemented, and u, v and w denote the left input, right input and output wires, respectively. We use $V_0(w)$ (resp., $V_1(w)$) as a short-hand for $V_0(\mathsf{C}_0, x_0, w)$ (resp., $V_1(\mathsf{C}_1, x_1, w)$), the function that returns the *value* of the wire w when the circuit C_0 (resp., C_1) is evaluated on the input x_0 (resp., x_1).

6.2.2 Garbling

The formal definition of syntax and security of garbling schemes is originally from [BHR12b]. Our definitions are taken mostly from [JSW17].

Definition 42. A garbling scheme GC is a tuple of PPT algorithms $(\mathsf{GCircuit}, \mathsf{GInput}, \mathsf{GEval})$ with syntax and semantics defined as follows.

- $(\widetilde{\mathsf{C}}, \mathtt{K}) \leftarrow \mathsf{GCircuit}(1^\lambda, \mathsf{C})$: On inputs a security parameter λ and a circuit $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$, the *garble-circuit* algorithm GCircuit outputs the *garbled circuit* $\widetilde{\mathsf{C}}$ and *key* $\mathtt{K}$.
- $\widetilde{x} \leftarrow \mathsf{GInput}(\mathtt{K}, x)$: On input an input $x \in \{0,1\}^n$ and key $\mathtt{K}$, the *garble-input* algorithm GInput outputs $\widetilde{x}$.
- $y = \mathsf{GEval}(\widetilde{\mathsf{C}}, \widetilde{x})$: On input a garbled circuit $\widetilde{\mathsf{C}}$ and a garbled input $\widetilde{x}$, the *evaluate* algorithm GEval outputs $y \in \{0,1\}^m$.

Correctness. There is a negligible function $\epsilon = \epsilon(\lambda)$ such that for any $\lambda \in \mathbb{N}$, any circuit C and input x it holds that

$$\Pr[\mathsf{C}(x) = \mathsf{GEval}(\widetilde{\mathsf{C}}, \widetilde{x})] = 1 - \epsilon(\lambda),$$

where $(\widetilde{\mathsf{C}}, \mathtt{K}) \leftarrow \mathsf{GCircuit}(1^\lambda, \mathsf{C})$, $\widetilde{x} \leftarrow \mathsf{GInput}(\mathtt{K}, x)$.

In this work we only consider the security notion of adaptive indistinguishability, which is strictly weaker than the usually considered notion of adaptive simulatability.

Definition 43 (Adaptive indistinguishability.)**.** A garbling scheme GC is (t, ϵ)-adaptively-indistinguishable for a class of circuits $\mathcal{C}$, if for any probabilistic adversary A running in time $t = t(\lambda)$,

$$\left|\Pr[\mathsf{G}_{\mathsf{A},\mathsf{GC}}(1^\lambda, 0) = 1] - \Pr[\mathsf{G}_{\mathsf{A},\mathsf{GC}}(1^\lambda, 1) = 1]\right| \leq \epsilon(\lambda).$$

where the experiment $\mathsf{G}_{\mathsf{A},\mathsf{GC},\mathsf{S}}(1^\lambda, b)$ is defined as follows:

1. A selects two circuits $\mathsf{C}_0, \mathsf{C}_1 \in \mathcal{C}$ such that $\Phi(\mathsf{C}_0) = \Phi(\mathsf{C}_1)$ and receives $\widetilde{\mathsf{C}}_b$ where $(\widetilde{\mathsf{C}}_b, \mathtt{K}) \leftarrow \mathsf{GCircuit}(1^\lambda, \mathsf{C}_b)$.
2. A specifies x_0, x_1 such that $\mathsf{C}_0(x_0) = \mathsf{C}_1(x_1)$ and receives $\widetilde{x}_b \leftarrow \mathsf{GInput}(\mathtt{K}, x_b)$.
3. Finally, A outputs a bit b', which is the output of the experiment.

For completeness, we also provide the definition of adaptive simulatability.

Definition 44 (Adaptive simulatability.)**.** A garbling scheme GC is (t, ϵ)-adaptively-simulatable for a class of circuits $\mathcal{C}$, if there exists a PPT time simulator $\mathsf{S} = (\mathsf{SCircuit}, \mathsf{SInput})$ such that, for any probabilistic adversary A running in time $t = t(\lambda)$,

$$\left|\Pr[\mathsf{F}_{\mathsf{A},\mathsf{GC},\mathsf{S}}(1^\lambda, 0) = 1] - \Pr[\mathsf{F}_{\mathsf{A},\mathsf{GC},\mathsf{S}}(1^\lambda, 1) = 1]\right| \leq \epsilon(\lambda).$$

where the experiment $\mathsf{F}_{\mathsf{A},\mathsf{GC},\mathsf{S}}(1^\lambda, b)$ is defined as follows:

1. The adversary A specifies $\mathsf{C} \in \mathcal{C}$ and gets $\widetilde{\mathsf{C}}$ created as follows:
   - if $b = 0$: $(\widetilde{\mathsf{C}}, \mathtt{K}) \leftarrow \mathsf{GCircuit}(1^\lambda, \mathsf{C})$,
   - if $b = 1$: $(\widetilde{\mathsf{C}}, z) \leftarrow \mathsf{SCircuit}(1^\lambda, \Phi(\mathsf{C}))$.
2. The adversary A specifies x and gets $\widetilde{x}$ created as follows:
   - if $b = 0$, $\widetilde{x} \leftarrow \mathsf{GInput}(\mathtt{K}, x)$,
   - if $b = 1$, $\widetilde{x} \leftarrow \mathsf{SInput}(\mathsf{C}(x), z)$.
3. Finally, the adversary outputs a bit b', which is the output of the experiment.

In the selective counterparts of Definitions 43 and 44, the adversary has to select (along with the circuit) the input also in the first step.

Remark 4. A few remarks concerning Definitions 43 and 44 are in order:

1. We call the experiments corresponding to $b = 0$ and $b = 1$ in Definition 44 "real" and "simulated" experiments, respectively. We call the experiments corresponding to $b = 0$ and $b = 1$ in Definition 43 the "left" and "right" experiments, respectively.
2. When the context is clear, we use the simpler notion F^0 and F^1 to denote the experiments $\mathsf{F}_{\mathsf{A},\mathsf{GC},\mathsf{S}}(1^\lambda, 0)$ and $\mathsf{F}_{\mathsf{A},\mathsf{GC},\mathsf{S}}(1^\lambda, 1)$, respectively. Similarly, we use G^0 and G^1 for the experiments in Definition 43.
3. We use $t_{\mathsf{G}} = t_{\mathsf{G}}(\lambda)$ to denote the time taken to run experiment G.

Offline Yao. In Algorithm 6.1 we describe Yao's original garbling scheme $\mathsf{YGC}_{\mathsf{SKE}}$ for Boolean circuits with fan-in 2, where the output map is sent along with the garbled circuit in the offline phase. Recall that in the variant from [JW16], the output map is sent along with the *garbled input* in the online phase. In addition to satisfying the standard notion of security for SKE scheme (IND-CPA), the SKE needs to satisfy the following property for correctness of the garbling schemes to hold.

Definition 45 (Special correctness [JW16])**.** We say that an SKE $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$ satisfies special correctness if for every security parameter λ, every key $\mathtt{k} \leftarrow \mathsf{Gen}(1^\lambda)$, every message $\mathtt{m} \in \mathcal{M}$, and encryption $\mathtt{c} \leftarrow \mathsf{Enc}_{\mathtt{k}}(\mathtt{m})$, it holds $\mathsf{Dec}_{\mathtt{k}'}(\mathtt{c}) = \perp$ for all $\mathtt{k}' \neq \mathtt{k}$ with overwhelming probability.

Algorithm 6.1: Offline Yao scheme $\mathsf{YGC}_{\mathsf{SKE}}$ based on a symmetric-key encryption scheme $\mathsf{SKE} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

$\mathsf{GCircuit}(1^\lambda, \mathsf{C})$
input:
1. Security parameter λ in unary
2. Circuit $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$ with N gates and W wires

1 **for** $j \in [1, W]$ *and* $\gamma \in \{0,1\}$ **do**
2 $\quad \mathtt{k}_j^\gamma \leftarrow \mathsf{Gen}(1^\lambda)$ `// Sample wire keys`
3 Set $\mu = \left\{(\mathtt{k}_j^0 \to 0, \mathtt{k}_j^1 \to 1)\right\}_{j \in [W-m+1, W]}$ `// Set output map`
4 **for** $j \in [n+1, N-m]$ **do** `// Garble circuit`
5 $\quad$ Let (g_j, u, v, w) denote the j-th gate in C
6 $\quad \widetilde{g}_j \leftarrow \mathsf{Real}(g_j, \{\mathtt{k}_u^\gamma, \mathtt{k}_v^\gamma, \mathtt{k}_w^\gamma\}_{\gamma \in \{0,1\}})$ `// Real defined in Table 6.3`
7 **return** $((\widetilde{\mathsf{C}}, \mu), \mathtt{K} = \mathbf{k})$, where $\widetilde{\mathsf{C}} := \{\widetilde{g}_j\}_{j \in [n+1, N-m]}$ and $\mathbf{k} := \left\{(\mathtt{k}_j^0, \mathtt{k}_j^1)\right\}_{j \in [1,n]}$

$\mathsf{GInput}(\mathtt{K}, x)$
input:
1. Garbling key $\mathtt{K}$ parsed as $\mathbf{k}$ as in Line 7
2. Input $x \in \{0,1\}^n$ with bit decomposition $\{x_1, \ldots, x_n\}$

8 Set $\mathbf{k}_x := \left\{\mathtt{k}_j^{x_j}\right\}_{j \in [1,n]}$ `// Select input keys from` $\mathbf{k}$
9 **return** $\widetilde{x} := \mathbf{k}_x$

$\mathsf{GEval}((\widetilde{\mathsf{C}}, \mu), \widetilde{x})$
input:
1. Garbled circuit $\widetilde{\mathsf{C}}$ parsed as $\{\widetilde{g}_j\}_{j \in [n+1, N-m]}$, output map μ
2. Garbled input $\widetilde{x} = \mathbf{k}_x$

Parse $\boldsymbol{k}_x$ as $\{\mathtt{k}_1, \ldots, \mathtt{k}_n\}$
10 **for** $j \in [n+1, N-m]$ **do** `// Decode circuit`
$\quad$ Let (u, v, w) denote the wires of g_j `// The topology is assumed public`
11 $\quad$ Parse $\widetilde{g}_j$ as $(\{\mathtt{c}_1, \ldots, \mathtt{c}_4\})$ `// Parse the gate table`
12 $\quad$ **for** $l \in [1, 4]$ **do** `// Decrypt each double ciphertext till successful`
13 $\quad\quad$ Let $\mathtt{m} := \mathsf{Dec}_{\mathtt{k}_u}(\mathsf{Dec}_{\mathtt{k}_v}(\mathtt{c}_l))$
14 $\quad\quad$ **if** $m \neq \perp$ **then** `// Use special correctness: Definition 45`
15 $\quad\quad\quad$ set $\mathtt{k}_w := \mathtt{m}$
Parse μ as $\left\{(\mathtt{k}_j^0 \to 0, \mathtt{k}_j^1 \to 1)\right\}_{j \in [W-m+1, W]}$
for $j \in [W-m+1, W]$ **do** `// Decode output`
16 $\quad$ Set $y_j = 0$ if $\mathtt{k}_j = \mathtt{k}_j^0$; else set $y_j = 1$
17 **return** y

6.2.3 Pebbling game

We recall the pebbling game that was used in [HJO+16, JW16] to argue adaptive security of variants of Yao's scheme.

Definition 46 (Black-grey (BG) pebbling game [HJO+16, JW16])**.** Consider a DAG $G = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = [1, N]$ and let $\mathcal{X}_{BG} = \{\perp, \mathtt{B}, \mathtt{G}\}$ denote the set of colours of the pebbles. Consider a sequence $\boldsymbol{\mathcal{P}} := (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ of pebbling configurations for G, where $\mathcal{P}_i \in \mathcal{X}_{BG}^{\mathcal{V}}$ for all

$i \in [0, \tau]$. We call such a sequence *a black-grey pebbling strategy* for G if (i) every vertex is empty in the initial configuration (i.e., $\mathcal{P}_0 = (\bot, \ldots, \bot)$), (ii) every vertex is grey-pebbled in the final configuration (i.e., $\mathcal{P}_\tau = (\mathtt{G}, \ldots, \mathtt{G})$) and (iii) every configuration is obtained by applying one of the following rules to its preceding configuration (see Figure 6.2):

1. $\bot \mapsto \mathtt{B}$: a black pebble can be placed on a vertex if its predecessors are black-pebbled, i.e., exists $j^* \in \mathcal{V}$ such that $\mathcal{P}_{i+1}(j^*) = \mathtt{B}$, $\mathcal{P}_i(j^*) = \bot$, $\mathcal{P}_i(j) = \mathtt{B}$ for all $j \in \mathsf{pre}_G(j^*)$, and $\mathcal{P}_{i+1}(j) = \mathcal{P}_i(j)$ for all $j \in \mathcal{V} \setminus \{j^*\}$.
2. $\mathtt{B} \mapsto \bot$: a black pebble can be removed from a vertex if its predecessors are black-pebbled, i.e., exists $j^* \in \mathcal{V}$ such that $\mathcal{P}_{i+1}(j^*) = \bot$, $\mathcal{P}_i(j^*) = \mathtt{B}$, $\mathcal{P}_i(j) = \mathtt{B}$ for all $j \in \mathsf{pre}_G(j^*)$, and $\mathcal{P}_{i+1}(j) = \mathcal{P}_i(j)$ for all $j \in \mathcal{V} \setminus \{j^*\}$.
3. $\mathtt{B} \mapsto \mathtt{G}$: a black pebble on a vertex $v \in \mathcal{V}$ can be replaced with a grey pebble if v's *successors* are pebbled (either black or grey), i.e., exists $j^* \in \mathcal{V}$ such that $\mathcal{P}_{i+1}(j^*) = \mathtt{G}$, $\mathcal{P}_i(j^*) = \mathtt{B}$, $\mathcal{P}_i(j) \in \{\mathtt{B}, \mathtt{G}\}$ for all $j \in \mathsf{suc}_G(j^*)$, and $\mathcal{P}_{i+1}(j) = \mathcal{P}_i(j)$ for all $j \in \mathcal{V} \setminus \{j^*\}$.

The *space-complexity* of a BG pebbling strategy $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ for a DAG G is defined as the maximum number of black pebbles used at any point in the strategy:

$$\sigma_G(\boldsymbol{\mathcal{P}}) := \max_{i \in [0,\tau]} |\{j \in [1, N] : \mathcal{P}_i(j) = \mathtt{B}\}|.$$

Definition 47. If $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ is a BG pebbling strategy of space-complexity σ for a graph G, we say that $\boldsymbol{\mathcal{P}}$ is a (σ, τ)-BG pebbling strategy for G. We say that a class of graphs $\mathcal{G}$ has a (σ, τ)-BG pebbling strategy if every graph $G \in \mathcal{G}$ has a (σ, τ)-BG pebbling strategy. Similarly, we say that a class of circuits $\mathcal{C}$ has a (σ, τ)-BG pebbling strategy if for every circuit $\mathsf{C} \in \mathcal{C}$, the underlying graph $\Phi(\mathsf{C})$ has a (σ, τ)-strategy.

Similar definitions apply to all other pebble games considered in this section.

Remark 5. The rule to place (Rule 46.1) or remove (Rule 46.2) a black pebble in Definition 46 is the same as in the original reversible node-pebbling game from [Ben89] (Definition 54); in the following we refer to the latter as reversible (black) pebbling (RB). Therefore the BG pebbling game can be thought of as an extension of the RB pebbling game (with a different goal).

6.2.4 Treewidth and separators

We recall the definition of treewidth and graph separators, and then state a crucial theorem connecting them, which will be exploited in our pebbling strategy. We emphasise that understanding the definition of treewidth is *not* essential to understanding our pebbling strategies: it is the notion of separators, along with Theorem 19, which is key.

Definition 48 ([RS86, Bod98])**.** A tree decomposition of a graph[11] $G = (\mathcal{V}, \mathcal{E})$ is a tree, T, with nodes $\mathcal{X}_1, \ldots, \mathcal{X}_p$, where each $\mathcal{X}_i \subseteq \mathcal{V}$, satisfying the following properties:

1. Each graph vertex is contained in at least one tree node (i.e., $\cup_{i \in [1,p]} \mathcal{X}_i = \mathcal{V}$).

[11] The notion of tree decomposition is usually considered for undirected graphs, however, the notion will also turn out useful in our setting, where we consider directed graphs.

2. For every edge $(v, w) \in \mathcal{E}$, there exists a node $\mathcal{X}_i$ that contains both v and w.

3. The tree nodes containing a vertex v form a connected subtree of T.

The width of a tree decomposition is the size of its largest node $\mathcal{X}_i$ minus one. Its *treewidth* $w(G)$ is the minimum width among all possible tree decompositions.

Definition 49 ([RS86])**.** For a graph $G = (\mathcal{V}, \mathcal{E})$, a set $\mathcal{S} \subseteq \mathcal{V}$ is said to be a separator if the graph $G_{|\mathcal{V}\setminus\mathcal{S}}$ has at least two components, and each of these components has size at most $2|\mathcal{V}|/3$.[12]

Theorem 19 ([RS86, Bod98])**.** *A graph G with treewidth $w(G)$ has a separator of size at most $w(G)$.*

6.3 Hybrid Argument and the BGR Pebbling Game

In this section, we formally show that black-grey-red (BGR) pebbling strategies lead to security reductions for Offline Yao. We start off in Section 6.3.1 by formally defining the BGR pebbling game and then explain the semantics of its pebbles, described already (albeit informally) in Section 6.1.2. This enables us to define a hybrid $\mathsf{H}_{\mathcal{P}}$ in terms of a pebbling configuration $\mathcal{P}$ (Algorithm 6.2). Then, in Section 6.3.2, we justify the pebbling rules by proving that neighbouring pebbling configurations can indeed be proven indistinguishable (Lemma 25). Finally, we put these two steps together in Section 6.3.3 and show that BGR strategies imply adaptive indistinguishability of Offline Yao (Theorem 22) using the piecewise-guessing framework from Chapter 3. Since most of the ideas in Sections 6.3.2 and 6.3.3 are similar to pre-existing works [JW16] and Chapter 3, we skip detailed proofs and resort to high-level sketches.

6.3.1 Pebbling Configurations and Hybrids

The BGR pebbling game is a symmetric version of the BG pebbling game. In addition to the ones in BG pebbling game (Rules 46.1 through 46.3), there are additional rules (Rules 50.4 and 50.5) which govern how the red pebbles interact with the grey pebbles. Intuitively speaking, the dynamics between no pebbles and black pebble (Rules 50.1 and 50.2) are similar to the dynamics between red pebbles and grey pebbles (Rule 50.5): see Remark 6. A more formal definition of the game is given next.

Definition 50 (Black-grey-Red (BGR) pebbling game)**.** Consider a DAG $G = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = [1, N]$ and let $\mathcal{X}_{BGR} = \{\bot, \mathtt{B}, \mathtt{G}, \mathtt{R}\}$ denote the set of colours of the pebbles. A pebble is called *greyscale* if it is black or grey. Consider a sequence $\boldsymbol{\mathcal{P}} := (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ of pebbling configurations for G, where $\mathcal{P}_i \in \mathcal{X}_{BGR}^{\mathcal{V}}$ for all $i \in [0, \tau]$. We call such a sequence *a BGR pebbling strategy* for G if (i) every vertex is empty in the initial configuration (i.e., $\mathcal{P}_0 = (\bot, \ldots, \bot)$), (ii) and every vertex is red-pebbled in the final configuration (i.e., $\mathcal{P}_\tau = (\mathtt{R}, \ldots, \mathtt{R})$) and (iii) every configuration is obtained by applying one of the following rules to its preceding configuration (see Figure 6.2):

[12] To be precise, such a separator is called "balanced" [GJ16]. In this paper, we only consider balanced separators.

1. $\bot \mapsto \mathtt{B}$: a black pebble can be placed on an empty vertex if its predecessors are black-pebbled, i.e., exists $j^* \in \mathcal{V}$ such that $\mathcal{P}_{i+1}(j^*) = \mathtt{B}$, $\mathcal{P}_i(j^*) = \bot$, $\mathcal{P}_i(j) = \mathtt{B}$ for all $j \in \mathsf{pre}_G(j^*)$, and $\mathcal{P}_{i+1}(j) = \mathcal{P}_i(j)$ for all $j \in \mathcal{V} \setminus \{j^*\}$.

2. $\mathtt{B} \mapsto \bot$: a black pebble can be removed from a vertex if its predecessors are black-pebbled, i.e., exists $j^* \in \mathcal{V}$ such that $\mathcal{P}_{i+1}(j^*) = \bot$, $\mathcal{P}_i(j^*) = \mathtt{B}$, $\mathcal{P}_i(j) = \mathtt{B}$ for all $j \in \mathsf{pre}_G(j^*)$, and $\mathcal{P}_{i+1}(j) = \mathcal{P}_i(j)$ for all $j \in \mathcal{V} \setminus \{j^*\}$.

3. $\mathtt{B} \mapsto \mathtt{G}$: a black pebble on a vertex $v \in \mathcal{V}$ can be replaced with a grey pebble if v's *successors* carry greyscale pebbles (i.e., either black or grey), i.e., exists $j^* \in \mathcal{V}$ such that $\mathcal{P}_{i+1}(j^*) = \mathtt{G}$, $\mathcal{P}_i(j^*) = \mathtt{B}$, $\mathcal{P}_i(j) \in \{\mathtt{B}, \mathtt{G}\}$ for all $j \in \mathsf{suc}_G(j^*)$, and $\mathcal{P}_{i+1}(j) = \mathcal{P}_i(j)$ for all $j \in \mathcal{V} \setminus \{j^*\}$.

4. $\mathtt{G} \mapsto \mathtt{B}$: a grey pebble on a vertex $v \in \mathcal{V}$ can be replaced with a black pebble if v's *successors* carry greyscale pebbles, i.e., exists $j^* \in \mathcal{V}$ such that $\mathcal{P}_{i+1}(j^*) = \mathtt{B}$, $\mathcal{P}_i(j^*) = \mathtt{G}$, $\mathcal{P}_i(j) \in \{\mathtt{B}, \mathtt{G}\}$ for all $j \in \mathsf{suc}_G(j^*)$, and $\mathcal{P}_{i+1}(j) = \mathcal{P}_i(j)$ for all $j \in \mathcal{V} \setminus \{j^*\}$.

5. $\mathtt{G} \leftrightarrow \mathtt{R}$: a grey pebble can be swapped with a red pebble if its predecessors are grey-pebbled, i.e., exists $j^* \in \mathcal{V}$ such that $\mathcal{P}_{i+1}(j^*), \mathcal{P}_i(j^*) \in \{\mathtt{G}, \mathtt{R}\}$, $\mathcal{P}_{i+1}(j^*) \neq \mathcal{P}_i(j^*)$, $\mathcal{P}_i(j) = \mathtt{G}$ for all $j \in \mathsf{pre}_G(j^*)$, and $\mathcal{P}_{i+1}(j) = \mathcal{P}_i(j)$ for all $j \in \mathcal{V} \setminus \{j^*\}$.

The *space-complexity* of a BGR pebbling strategy $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ for a DAG G is defined as the maximum number of *greyscale* pebbles used at any point in the strategy:

$$\sigma_G(\boldsymbol{\mathcal{P}}) := \max_{i \in [0,\tau]} |\{j \in [1, N] : \mathcal{P}_i(j) \in \{\mathtt{B}, \mathtt{G}\}\}|.$$

Remark 6. A few remarks on the BGR pebbling game are in order:

1. Note that Rules 50.1 through 50.3 correspond to Rules 46.1 through 46.3 in the BG pebbling game. The end goals in the two games are however different.

2. When restricted to either black and empty (Rules 50.1 and 50.2) or grey and red pebbles (Rule 50.5), the BGR pebbling game simplifies to the reversible black pebbling game of Bennett [Ben89] defined in Definition 54. This is obvious for the black pebbles since the BGR pebbling game is an extension of the BG pebbling game which, in turn, is an extension of the reversible black pebbling game. To see why this is the case for grey and red pebbles, simply think of vertices with red pebbles as being empty (i.e., $\mathtt{R} = \bot$) and grey pebbles as black pebbles (i.e., $\mathtt{G} = \mathtt{B}$), and note that Rule 50.5 is now the same as Rules 50.1 and 50.2. Therefore, if one starts with an all-red (i.e., empty) configuration, the grey pebbles can be placed using reversible pebbling rules. Some of the reversible pebbling strategies will serve as crucial subroutines in the BGR pebbling strategies in the coming sections.

3. When restricted to black and grey pebbles, the BGR pebbling game again simplifies to the reversible pebbling game played on the graph with the direction of the edges *flipped*. However, we do not make use of this observation.

4. Only black pebbles can be placed on empty vertices. grey (resp., red) pebbles have to replace black or red (resp., grey) pebbles, respectively.

	Real ($\perp$)	Input ($\mathtt{B}$)	Sim ($\mathtt{G}$)
$c_{0,0}$	$\mathsf{Enc}_{k_u^0}(\mathsf{Enc}_{k_v^0}(k_w^{g(0,0)}))$	$\mathsf{Enc}_{k_u^0}(\mathsf{Enc}_{k_v^0}(k_w^{V(w)}))$	$\mathsf{Enc}_{k_u^0}(\mathsf{Enc}_{k_v^0}(k_w^{0}))$
$c_{0,1}$	$\mathsf{Enc}_{k_u^0}(\mathsf{Enc}_{k_v^1}(k_w^{g(0,1)}))$	$\mathsf{Enc}_{k_u^0}(\mathsf{Enc}_{k_v^1}(k_w^{V(w)}))$	$\mathsf{Enc}_{k_u^0}(\mathsf{Enc}_{k_v^1}(k_w^{0}))$
$c_{1,0}$	$\mathsf{Enc}_{k_u^1}(\mathsf{Enc}_{k_v^0}(k_w^{g(1,0)}))$	$\mathsf{Enc}_{k_u^1}(\mathsf{Enc}_{k_v^0}(k_w^{V(w)}))$	$\mathsf{Enc}_{k_u^1}(\mathsf{Enc}_{k_v^0}(k_w^{0}))$
$c_{1,1}$	$\mathsf{Enc}_{k_u^1}(\mathsf{Enc}_{k_v^1}(k_w^{g(1,1)}))$	$\mathsf{Enc}_{k_u^1}(\mathsf{Enc}_{k_v^1}(k_w^{V(w)}))$	$\mathsf{Enc}_{k_u^1}(\mathsf{Enc}_{k_v^1}(k_w^{0}))$

(a)

	Real_0 ($\perp$)	Input_0 ($\mathtt{B}$)	Input_1 ($\mathtt{G}$)	Real_1 ($\mathtt{R}$)
$c_{0,0}$	$\mathsf{Enc}_{k_u^0}(\mathsf{Enc}_{k_v^0}(k_w^{g_0(0,0)}))$	$\mathsf{Enc}_{k_u^0}(\mathsf{Enc}_{k_v^0}(k_w^{V_0(w)}))$	$\mathsf{Enc}_{k_u^0}(\mathsf{Enc}_{k_v^0}(k_w^{V_1(w)}))$	$\mathsf{Enc}_{k_u^0}(\mathsf{Enc}_{k_v^0}(k_w^{g_1(0,0)}))$
$c_{0,1}$	$\mathsf{Enc}_{k_u^0}(\mathsf{Enc}_{k_v^1}(k_w^{g_0(0,1)}))$	$\mathsf{Enc}_{k_u^0}(\mathsf{Enc}_{k_v^1}(k_w^{V_0(w)}))$	$\mathsf{Enc}_{k_u^0}(\mathsf{Enc}_{k_v^1}(k_w^{V_1(w)}))$	$\mathsf{Enc}_{k_u^0}(\mathsf{Enc}_{k_v^1}(k_w^{g_1(0,1)}))$
$c_{1,0}$	$\mathsf{Enc}_{k_u^1}(\mathsf{Enc}_{k_v^0}(k_w^{g_0(1,0)}))$	$\mathsf{Enc}_{k_u^1}(\mathsf{Enc}_{k_v^0}(k_w^{V_0(w)}))$	$\mathsf{Enc}_{k_u^1}(\mathsf{Enc}_{k_v^0}(k_w^{V_1(w)}))$	$\mathsf{Enc}_{k_u^1}(\mathsf{Enc}_{k_v^0}(k_w^{g_1(1,0)}))$
$c_{1,1}$	$\mathsf{Enc}_{k_u^1}(\mathsf{Enc}_{k_v^1}(k_w^{g_0(1,1)}))$	$\mathsf{Enc}_{k_u^1}(\mathsf{Enc}_{k_v^1}(k_w^{V_0(w)}))$	$\mathsf{Enc}_{k_u^1}(\mathsf{Enc}_{k_v^1}(k_w^{V_1(w)}))$	$\mathsf{Enc}_{k_u^1}(\mathsf{Enc}_{k_v^1}(k_w^{g_1(1,1)}))$

(b)

Table 6.3: (a) Garbling modes in [JW16]. The gate is denoted by g and the value of its output wire w when run on input x is denoted by $V(w)$. (b) Garbling modes in our case. The gates g_0 and g_1 are the gates in the same position in the circuits C_0 and C_1, respectively. The value $V_0(w)$ (resp., $V_1(w)$) denotes the bit going over the wire w in the computation $\mathsf{C}_0(x_0)$ (resp., $\mathsf{C}_1(x_1)$).

5. By the pebbling rules, in any strategy a vertex that is empty can *never* end up adjacent to another vertex with a red pebble in any BGR pebbling strategy. Moreover, a vertex with a grey (resp., black) pebble cannot be a predecessor of a vertex with no (resp. a red) pebble; the converse is however possible. These properties will turn out to be important sanity checks in ensuring the validity of BGR pebbling strategies in the later sections. Moreover, they ensure correctness of the simulations they represent.

Template for Hybrids.

A pebbling configuration $\mathcal{P} \in \mathcal{X}^{\mathcal{V}}_{BGR}$ is used to encode a selective hybrid $\mathsf{H}_{\mathcal{P}}$. For an *internal gate* v, the translation is carried out as described below:

- if v carries no pebble ($\perp$) in $\mathcal{P}$ then g is garbled as in the left game (Real_0),
- a black pebble ($\mathtt{B}$) on v indicates that the garbling of g is input-dependant on x_0 and C_0, (Input_0)
- a grey pebble ($\mathtt{G}$) on v indicates that the garbling of g is input-dependant on x_1 and C_1 (Input_1)
- a red pebble ($\mathtt{R}$) on v indicates g is garbled as in the right game (Real_1).

The distributions corresponding to the four garbling modes – Real_0, Input_0, Input_1 and Real_1 – are formally defined in Table 6.3(b). (Note that the semantics of grey pebbles is different from that in the BG pebbling game.) This information is sufficient to construct the garbled circuit $\widetilde{\mathsf{C}}$. What remains to complete the description of $\mathsf{H}_{\mathcal{P}}$, is describing how to generate the input garbling $\widetilde{x}$ and the output map. If an input gate carries no (resp., a red) pebble then the

Algorithm 6.2: Selective hybrids: the adversary commits to the garbling inputs at the beginning of the game (boxed).

$\mathsf{H}_{\mathcal{P}}(\lambda)$

parameters:

1. Circuit parameters N (gates), W (wires), n and m (input and output size)
2. Selective adversary A
3. Symmetric encryption scheme SKE $:=$ (Gen, Enc, Dec)

index: A BGR pebbling configuration $\mathcal{P} \in \mathcal{X}_{BGR}^{[1,N]}$

Receive circuits C_0 and C_1 from A // $\mathsf{H} \leftarrow \mathsf{A}$

Receive inputs $x_0, x_1 \in \{0,1\}^n$ as commitment from A // $\mathsf{H} \leftarrow \mathsf{A}$

for $j \in [1, W]$ *and* $\gamma \in \{0,1\}$ **do**
  $\mathrm{k}_j^\gamma \leftarrow \mathsf{Gen}(1^\lambda)$ // Sample wire keys

Set $\mu = \left\{(\mathrm{k}_j^0 \to 0, \mathrm{k}_j^1 \to 1)\right\}_{j \in [W-m+1, W]}$ // Set output map

for $j \in [n+1, N-m]$ // Garble circuit

do
  For $b \in \{0,1\}$, let the j-th gate in C_b be $(g_{b,j}, u, v, w)$
  if $\mathcal{P}(n+j) = \perp$ **then**
    $\widetilde{g}_j \leftarrow \mathsf{Real}_0(g_{0,j}, \{\mathrm{k}_u^\gamma, \mathrm{k}_v^\gamma, \mathrm{k}_w^\gamma\}_{\gamma \in \{0,1\}})$ // No pebble
  else if $\mathcal{P}(n+j) = B$ **then**
    $\widetilde{g}_j \leftarrow \mathsf{Input}_0(g_{0,j}, \{\mathrm{k}_u^\gamma, \mathrm{k}_v^\gamma\}_{\gamma \in \{0,1\}}, \mathrm{k}_w^{V_0(w)})$ // B-pebbled
  else if $\mathcal{P}(n+j) = G$ **then**
    $\widetilde{g}_j \leftarrow \mathsf{Input}_1(g_{1,j}, \{\mathrm{k}_u^\gamma, \mathrm{k}_v^\gamma\}_{\gamma \in \{0,1\}}, \mathrm{k}_w^{V_1(w)})$ // G-pebbled
  else
    $\widetilde{g}_j \leftarrow \mathsf{Real}_1(g_{1,j}, \{\mathrm{k}_u^\gamma, \mathrm{k}_v^\gamma, \mathrm{k}_w^\gamma\}_{\gamma \in \{0,1\}})$ // R-pebbled

for $j \in [1, n]$ // Garble input

do
  if $\mathcal{P}(j) \in \{\perp, B\}$ **then**
    Set $\widetilde{x}_j = \mathrm{k}_j^{x_{0,j}}$
  else
    Set $\widetilde{x}_j = \mathrm{k}_j^{x_{1,j}}$

Send $(\widetilde{\mathsf{C}}, \mu, \widetilde{x})$ to A where $\widetilde{\mathsf{C}} := \{\widetilde{g}_j\}_{j \in [n+1, N-m]}$ and $\widetilde{x} := \{\widetilde{x}_j\}_{j \in [1,n]}$ // $\mathsf{H} \to \mathsf{A}$

Receive a bit b' from A // $\mathsf{H} \leftarrow \mathsf{A}$

return b'

garbling key for x_0 (resp., x_1) is selected in that hybrid. The output map, on the other hand, is simply the default one prescribed in the scheme and therefore the pebbles on the output gates are ignored. We refer the readers to Algorithm 6.2 for a formal definition of $\mathsf{H}_{\mathcal{P}}$.

Sequence of hybrids. A pebbling strategy $\boldsymbol{\mathcal{P}} = \{\mathcal{P}_0, \ldots, \mathcal{P}_\tau\}$ will give rise to a sequence of selective hybrids

$$\mathsf{H}^0 = \mathsf{H}_{\mathcal{P}_0}, \ldots, \mathsf{H}_{\mathcal{P}_\tau} = \mathsf{H}^1, \tag{6.1}$$

Note that the extreme games correspond to the left selective experiment $\mathsf{H}^0 = \mathsf{H}_{\mathsf{A},\mathsf{GC}}(1^\lambda, 0)$ (since $\mathcal{P}_0 = (\perp, \ldots, \perp)$) and right selective experiment $\mathsf{H}^1 = \mathsf{H}_{\mathsf{A},\mathsf{GC}}(1^\lambda, 1)$ (since $\mathcal{P}_\tau =$

$(\mathtt{R}, \ldots, \mathtt{R})$), respectively. The exact pebbling strategy will be discussed later in Section 6.4. In the next section, we prove the indistinguishability of two neighbouring hybrids in such a sequence.

6.3.2 Indistinguishability of Neighbouring Hybrids

We will now prove that any two neighbouring hybrids in such a sequence of hybrids that was derived from a BGR pebbling strategy are indeed indistinguishable.

Lemma 25 (neighbouring indistinguishability)**.** *Let $\mathcal{P}$ and $\mathcal{Q}$ denote two neighbouring configurations in a BGR pebbling strategy. If the underlying encryption scheme* SKE *is (t, ϵ)-IND-CPA secure, then $\mathsf{H}_{\mathcal{P}}$ and $\mathsf{H}_{\mathcal{Q}}$ (as defined in Algorithm 6.2) are $(t - t_{\mathsf{H}}, 3\epsilon)$-indistinguishable, i.e., for any adversary* A *running in time at most $t - t_{\mathsf{H}}$*

$$|\Pr\langle \mathsf{A}, \mathsf{H}_{\mathcal{P}}\rangle = 1 - \Pr\langle \mathsf{A}, \mathsf{H}_{\mathcal{Q}}\rangle = 1| \leq 3\epsilon.$$

Proof. Recall that hybrids correspond to pebbling configurations and that two neighbouring hybrids differ by a single pebble. We split the proof into three cases which correspond to the pebbling moves $\bot \leftrightarrow \mathtt{B}$, $\mathtt{B} \leftrightarrow \mathtt{G}$ and $\mathtt{R} \leftrightarrow \mathtt{G}$ respectively. The reduction in the first and last case is similar, and relies on the indistinguishability of the underlying encryption scheme (similar to [JW16, Lemma 1]). Therefore in the claim below we focus on the first case. In the second case, we argue that the hybrids are identically distributed (similar to [JW16, Lemma 2]). We only provide proof sketches here.

Claim 1 (Rules 50.1 and 50.2: $\bot \leftrightarrow \mathtt{B}$)**.** *If the underlying encryption scheme* SKE *is (t, ϵ)-IND-CPA secure, and if $\mathcal{Q}$ is obtained from $\mathcal{P}$ using Rule 50.1 or Rule 50.2 then the hybrids $\mathsf{H}_{\mathcal{P}}$ and $\mathsf{H}_{\mathcal{Q}}$ are $(t - t_{\mathsf{H}}, 3\epsilon)$-indistinguishable.*

Proof (Sketch). Let's consider the case where $\mathcal{Q}$ is obtained from $\mathcal{P}$ using Rule 50.1, i.e. $\mathcal{Q}$ is obtained by placing a black pebble on an empty vertex g^* in $\mathcal{P}$ – the complementary case of Rule 50.2 can be argued in a symmetric manner. Recall that, by the BGR pebbling rules, there are two possible cases:

1. g^* is a source vertex. In this case $\mathsf{H}_{\mathcal{P}}$ and $\mathsf{H}_{\mathcal{Q}}$ are identical experiments.
2. g^* is an internal vertex – let us denote the corresponding gate in C_0 by (g^*, u^*, v^*, w^*). In this case, the predecessors of g^* are guaranteed to be pebbled black in the configuration $\mathcal{P}$. By the semantics of the pebbles, this means that both the predecessors of g^* are in Input_0 mode in hybrids $\mathsf{H}_{\mathcal{P}}$ and $\mathsf{H}_{\mathcal{Q}}$. This, in turn, means that one of the output keys corresponding to the incoming wire u^* is not used in the garbling table of that predecessor gate; the same holds for the wire w^*. Therefore these keys are free and the reduction can set them as challenge key of the underlying encryption scheme in order to switch the garbling table of g^* from Real_0 to Input_0 in three steps. This corresponds to a black pebble on g^* and therefore the hybrid $\mathsf{H}_{\mathcal{Q}}$.

□

Claim 2 (Rules 50.3 and 50.4: $\mathtt{B} \leftrightarrow \mathtt{G}$)**.** *If $\mathcal{Q}$ is obtained from $\mathcal{P}$ using Rule 50.3 or Rule 50.4 then the hybrids $\mathsf{H}_{\mathcal{P}}$ and $\mathsf{H}_{\mathcal{Q}}$ are identically distributed.*

Proof (Sketch). Let's restrict to the case where $\mathcal{Q}$ is obtained from $\mathcal{P}$ using Rule 50.3, i.e., $\mathcal{Q}$ is obtained by replacing a black pebble on a vertex g^* in $\mathcal{P}$ with a grey pebble – as in the

previous proof the complementary case of Rule 50.4 is symmetric. By the BGR pebbling rules, there are two possible cases:

1. g^* is a sink vertex. Since $\mathsf{C}_0(x_0) = \mathsf{C}_1(x_1)$ and g^* is in input-dependant mode in both $\mathsf{H}_{\mathcal{P}}$ (Input_0) and $\mathsf{H}_{\mathcal{Q}}$ (Input_1) the simulations are identical.
2. g^* is an internal vertex – let us denote the corresponding gates in C_0 and C_1 by (g_0^*, u^*, v^*, w^*) and (g_1^*, u^*, v^*, w^*) respectively. This means that g^* is in Input_0 mode in the hybrid $\mathsf{H}_{\mathcal{P}}$ and Input_1 mode in the hybrid $\mathsf{H}_{\mathcal{Q}}$ (with all the successors in Input_0 or Input_1 mode in both the hybrids). In case that the value of the output wire w^* is the same for both hybrids (i.e., $V_0(w^*) = V_1(w^*)$) then we are done by the same argument as the case above in Item 1. So let's focus on the case where the values are different. Since both the keys corresponding to w^* are treated symmetrically in the successor gates, swapping them in the garbling table results in identical distributions and therefore $\mathsf{H}_{\mathcal{P}}$ and $\mathsf{H}_{\mathcal{Q}}$ are also identically distributed.

This proves the lemma. □

Selective indistinguishability. Combining Lemma 25 with the semantics of the pebbles (Table 6.3) yields (via the standard hybrid argument) selective indistinguishability of Offline Yao.

Theorem 20. *Suppose that a class of circuits $\mathcal{C}$ has a (σ, τ)-BGR pebbling strategy. If the encryption scheme* SKE *is (t, ϵ)-secure then* $\mathsf{YGC}_{\mathsf{SKE}}$ *is $(t - t_{\mathsf{H}}, 3\tau\epsilon)$-selectively-indistinguishable for $\mathcal{C}$.*

Remark 7. Theorem 20 was already proven in a stronger form in [LP09]: it was shown there that Offline Yao is selectively-*simulatable* under similar assumptions. Here it only serves as a stepping stone to the main theorem (Theorem 22) to be proven in the next section.

6.3.3 Adaptive Indistinguishability via Piecewise Guessing

Observe that in the hybrid $\mathsf{H}_{\mathcal{P}}$ described in Algorithm 6.2, the knowledge of the committed garbling inputs x_0 and x_1 (Algorithm 6.2) is used to compute the output value of gates that carry greyscale pebbles in the configuration $\mathcal{P}$ (Lines 9 and 11). So, in principle, the simulation of $\mathsf{H}_{\mathcal{P}}$ can be carried out if this information is available as an "advice". Moreover, the indistinguishability of two successive hybrids can be shown (Lemma 25) if such advice for *both* the hybrids is available. In case the number of greyscale pebbles is small, the size of this advice could potentially be smaller than the size of garbling inputs x_0 and x_1. This means that it is possible to apply the Piecewise-Guessing framework from Chapter 3. We explain this in detail next.

Applying the piecewise-guessing framework. The main theorem in Chapter 3 is stated below in Theorem 21 after having been simplified and tailored for our application to circuit garbing. The result of applying Theorem 21 to Offline Yao is stated in Theorem 22. Furthermore, exploiting the properties of the pebbling strategies we design, we provide an optimised version of Theorem 22 later in Section 6.4.3 (Theorem 24).

Theorem 21 (Theorem 2 in Chapter 3 tailored to Definition 43)**.** *Let G^0 and G^1 be as in Definition 43, H^0 and H^1 the selective indistinguishability games. Furthermore, let $\mathsf{H}^0 = \mathsf{H}_{\mathcal{P}_0}, \ldots, \mathsf{H}_{\mathcal{P}_\tau} = \mathsf{H}^1$ be the sequence of hybrids from Equation* (6.1) *and suppose that every*

pebbling configuration $\mathcal{P}_i$ in the strategy $\mathcal{P}_0, \ldots, \mathcal{P}_\tau$ can be be computed in time t_{P}. Assume that for each $i \in [0, \tau - 1]$, there exists a function $\alpha_i \colon \{0,1\}^* \to \{0,1\}^\sigma$ such that the hybrids $\mathsf{H}_{\mathcal{P}_i}$ and $\mathsf{H}_{\mathcal{P}_{i+1}}$ are (t, ϵ)-indistinguishable when A *commits to $\alpha_i(\mathsf{C}_0, \mathsf{C}_1, x_0, x_1)$ as advice at the beginning of the experiment (instead of (x_0, x_1)).*[13] *Then G^0 and G^1 are $(t - t_\sigma - t_{\mathsf{P}}, \tau \cdot 2^\sigma \cdot \epsilon)$-indistinguishable (where t_σ denotes the time to sample a string in $\{0,1\}^\sigma$ uniformly at random).*

Theorem 22. *Suppose that a class of circuits $\mathcal{C}$ has a (σ, τ)-BGR pebbling strategy. If the encryption scheme* SKE *is (t, ϵ)-secure then $\mathsf{YGC}^{\mathsf{SKE}}$ is $(t - t_{\mathsf{H}} - t_\sigma - t_{\mathsf{P}}, \tau 2^\sigma \cdot 3\epsilon)$-adaptively-indistinguishable for $\mathcal{C}$.*

Proof Sketch. As already observed, the advice function α_i should return the values of the output wires of all those gates that carry greyscale pebbles in $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$. Therefore, in Theorem 21 we set

$$\begin{aligned} \alpha_i(\mathsf{C}_0, \mathsf{C}_1, x_0, x_1) := (V_0(w) : (g, u, v, w) \in \mathsf{C}_0 \ \text{ and } \ \mathcal{P}(g) = \mathtt{B}) \| \\ (V_1(w) : (g, u, v, w) \in \mathsf{C}_1 \ \text{ and } \ \mathcal{P}(g) = \mathtt{G}) \end{aligned} \tag{6.2}$$

where $\mathcal{P} := \mathcal{P}_{i+1}$ if $\mathcal{P}_{i+1}$ is obtained from $\mathcal{P}_i$ by adding a greyscale pebble; and $\mathcal{P} := \mathcal{P}_i$ otherwise.[14]

The length of the advice is therefore smaller than in the selective hybrid in case the pebbling complexity of $G = \Phi(\mathsf{C}_0) = \Phi(\mathsf{C}_1)$ is smaller than the input length. What remains is to show that indistinguishability of two consecutive hybrids can be shown relying only on $\boldsymbol{\alpha}_i := \alpha_i(\mathsf{C}_0, \mathsf{C}_1, x_0, x_1)$. To see this note that the knowledge of the committed garbling inputs x_0 and x_1 (Algorithm 6.2) is used to compute the output value of gates that carry greyscale pebbles in the configuration $\mathcal{P}$ (Lines 9 and 11). Since these are already present in the hint, the reduction algorithm can simply extract these values from $\boldsymbol{\alpha}_i$ and use them instead of explicitly computing $V_0(\cdot)$ and $V_1(\cdot)$. Following the arguments in Lemma 25, we get that if the encryption scheme is (t, ϵ)-secure then the experiments $\mathsf{H}_{\mathcal{P}_i}$ and $\mathsf{H}_{\mathcal{P}_{i+1}}$ are $(t, 3\epsilon)$-indistinguishable when A commits to $\boldsymbol{\alpha}_i$, and the proof now follows Theorem 21.

□

6.4 BGR Pebbling Strategy

In this section, we describe our main strategy for the BGR pebbling game. As a warm up, we describe a BGR pebbling strategy (Section 6.4.1) for paths on $N \in \mathbb{N}$ vertices that uses $O(\log(N))$ greyscale pebbles. It is instructive to go through it as it serves the following two purposes.

1. Firstly, it highlights the fact that coming up with space-efficient BGR pebbling strategies is non-trivial.

[13] To show this formlly, we need to show that for every pair of neighbouring hybrids H_i and H_{i+1}, there exist *partially-selective* hybrids $\hat{\mathsf{H}}_{i,0}$ and $\hat{\mathsf{H}}_{i,1}$ and a function α_i such that (i) H_i and H_{i+1} are "partially-selectivised" versions of $\hat{\mathsf{H}}_{i,0}$ and $\hat{\mathsf{H}}_{i,1}$ using the function α_i and (ii) $\hat{\mathsf{H}}_{i,0}$ and $\hat{\mathsf{H}}_{i,1}$ are indistinguishable.

[14] Recall from the proof of Lemma 25 that for pebbling configurations $\mathcal{P}_i$ and $\mathcal{P}_{i+1}$ that differ by a pebbling move $\mathtt{B} \leftrightarrow \mathtt{G}$, the corresponding hybrids $\mathsf{H}_{\mathcal{P}_1}$ and $\mathsf{H}_{\mathcal{P}_{i+1}}$ are *identically* distributed.

2. More importantly, it serves as a stepping stone to the main strategy: paths can be thought of as graphs of (tree)width one and our main strategy (Section 6.4.2) extends this strategy using the notion of separators.

Finally, we discuss the implications of our pebbling strategy to the security of Offline Yao (Section 6.4.3).

6.4.1 Warm up: BGR Strategy for Paths

Before describing BGRPath, the BGR stategy for paths, we describe RBPath, Bennett's [Ben89] space-optimal RB pebbling strategy, as it is used as a sub-strategy in BGRPath. RBPath places a black pebble on the sink of a path of length $N = 2^n$ using $n + 1$ black pebbles (in $N^{\log(3)}$) steps). Since the strategy is reversible, by RBPath^{-1} we denote the reverse strategy that *removes* the black pebble on the sink. This strategy, thanks to the observation in Remark 6.2, will also be used to both place or remove a grey pebble on an all-red-pebbled graph.

Space-Optimal RB Pebbling.

For ease of presentation, let's assume that the path has $N = 2^n$ vertices $[1, N]$ – the strategy can easily be extended to paths of arbitrary length by splitting it up appropriately. The strategy RBPath_n is described recursive in n. Since the strategy is reversible, we denote its inverse by RBPath_n^{-1}.

For the case $n = 0$, RBPath_1 simply places a pebble on v^*.

For $n > 0$ and path P_{2^n}, let's refer to $P_{2^n|[1,2^{n-1}]}$ and $P_{2^n|[2^{n-1}+1,2^n]}$ as the first and second halves respectively. RBPath_n then does the following.

1. Recursively place a black pebble on the first half ($\mathsf{RBPath}_{n-1}(P_{2^n|[1,2^{n-1}]})$)
2. Recursively place a black pebble on the second half ($\mathsf{RBPath}_{n-1}(P_{2^n|[2^{n-1},2^n]})$)
3. Recursively remove the black pebble on the first half ($\mathsf{RBPath}_{n-1}^{-1}(P_{2^n|[1,2^{n-1}]})$)

Lemma 26. *P_{2^n} can be RB-pebbled using at most $\sigma := n + 1$ black pebbles in at most $\tau := 3^n = 2^{n\log(3)}$ steps.*

Proof. For the base case of $n = 0$, RBPath_0 simply places a pebble on v^*, hence requires 1 pebble and 1 step. Let's assume that RBPath_{n-1} can Dpebble $P_{2^{n-1}}$ using $\log(2^{n-1}) + 1 = n$ pebbles in 3^{n-1} steps. Now, to pebble the sink of P_{2^n}, RBPath_n first pebbles the sink of the first half, i.e., the midpoint, then leaves the midpoint pebbled and pebbles the second half, and finally keeps the sink pebbled while removing the pebble on the midpoint. Thus, RBPath_n requires at most $n + 1$ pebbles and takes at most $3 \cdot 3^{n-1} = 3^n$ steps. This proves the lemma. □

Lemma 27. *Starting from the all-red configuration, P_{2^n} can be grey-pebbled using at most $\sigma := n + 1$ grey pebbles in at most $\tau := 3^n = 2^{n\log(3)}$ steps.*

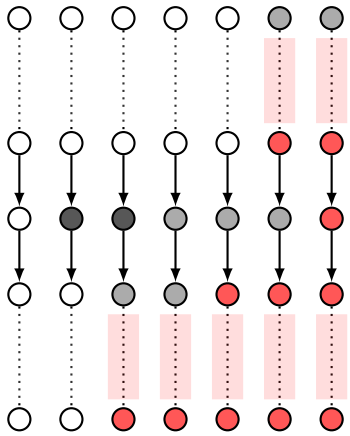


Figure 6.3: Recursive step in the BGR pebbling strategy BGRPath. We start from an empty configuration (leftmost path) and apply Item 1 through Item 6 to end up with the penultimate configuration (rightmost path).

Space-Optimal BGR Strategy.

Let's denote the pebbling strategy by BGRPath. For ease of presentation, we make two simplifying assumptions:

1. the path has $N = 2^n - 1$ vertices $[1, N]$ – the strategy can easily be extended to paths of arbitrary length; and
2. the final configuration produced by BGRPath is not the all-red configuration as prescribed in Definition 50 but the *penultimate* one with a grey pebble on the source and the rest pebbled red (i.e., $(\mathtt{G}, \mathtt{R}, \ldots, \mathtt{R})$) – getting to the all-red configuration from that is possible in one move by applying Rule 50.5.

The strategy BGRPath is described below recursively in n.

For the base case of $n = 2$, $\mathsf{BGRPath}_2$ works as follows:

1. Place a black pebble on 1, 2 and 3 (in that order) (Rule 50.1: $\bot \mapsto \mathtt{B}$)
2. Replace the black pebbles on 3, 2 and 1 (in that order) with a grey pebble (Rule 50.3: $\mathtt{B} \mapsto \mathtt{G}$)
3. Replace the grey pebble on 3 and 2 (in that order) with a red pebble (Rule 50.5: $\mathtt{G} \mapsto \mathtt{R}$)

For $n > 2$, let's refer to the vertex 2^{n-1} as the *midpoint* of P_N, the path with $N = 2^n - 1$ vertices. Moreover, let's refer to $P_{N|[1,2^{n-1}-1]}$ and $P_{N|[2^{n-1}+1,N]}$ as the first and second halves of P_N respectively. Then P_N can be switched from the empty configuration to the penultimate configuration as follows (see Figure 6.3).

1. Recursively place a black pebble on the midpoint 2^{n-1} ($\mathsf{RBPath}(P_{N|[1,2^{n-1}]})$)
2. Recursively switch the second half to its penultimate configuration ($\mathsf{BGRPath}_{n-1}(P_{N|[2^{n-1}+1,N]})$)
3. Replace the black pebble on the midpoint 2^{n-1} with a grey pebble (Rule 50.3)

4. Replace the grey pebble on $2^{n-1}+1$ with a red pebble (Rule 50.5)
5. Recursively switch the first half to its penultimate configuration ($\mathsf{BGRPath}_{n-1}(P_{N|[1,2^{n-1}-1]})$)
6. Recursively replace the grey pebble on the midpoint 2^{n-1} with a red pebble ($\mathsf{RGRPath}^{-1}(P_{N|[2,2^{n-1}]})$)

Lemma 28. *For $N = 2^n - 1$, P_N can be BGR-pebbled using at most $\sigma := n + 2$ greyscale pebbles in at most $\tau := 2^{n+1} \cdot 3^n = 2^{n(\log(3)+1)+1}$ steps.*

Proof. For the base case of $n = 2$, $\mathsf{BGRPath}_2$ requires at most $n + 2 = 3$ greyscale pebbles *and* a red pebble is *never* placed on the start vertex 1. Now, let's assume that on input $P_{2^{n-1}-1}$, the path with $2^{n-1} - 1$ vertices, $\mathsf{BGRPath}_{n-1}$

1. switches $P_{2^{n-1}-1}$ from the empty configuration to the penultimate configuration using $n - 1 + 2 = n + 1$ greyscale pebbles; and
2. never places a red pebble on the start vertex 1.

For the induction step, first note that the inductive step in Item 2 is only possible thanks to the black pebble left as a "marker" on 2^{n-1} in the previous step. This ensures that whenever a black pebble needs to be placed on $2^{n-1}+1$ this can be done thanks to the marker. Moreover, the induction hypothesis ensures that a red pebble never has to be placed on $2^{n-1}+1$ throughout the execution of $\mathsf{BGRPath}_{n-1}(P_{N|[2^{n-1}+1,N]})$ (since $2^{n-1}+1$ is the start vertex of the second half). Analogously, in the inductive step in Item 5 the (now) grey pebble on 2^{n-1} acts as the marker: whenever a black or grey pebble needs to be placed on $2^{n-1}-1$ this can be done thanks to the marker.
To complete the proof, we have to argue that (i) the number of greyscale pebbles used in $\mathsf{BGRPath}_n$ is one more than that required for the inductive step $\mathsf{BGRPath}_{n-1}$; (ii) the number of steps used in $\mathsf{BGRPath}_n$ is $2^{n+1} \cdot 3^n$; and (iii) the first vertex never carries a red pebble. To see why (i) is true, note that the number of pebbles required for $\mathsf{BGRPath}_n$ is governed by the recursion

$$\sigma(n) = \max(2, 1 + \sigma(n-1), n+2) \leq n + 2$$

where the $(n + 2)$ term is to account for the reversible pebbling carried out in Item 6 (while keeping the first node grey-pebbled), and the last inequality follows by the induction hypothesis. For (ii), write the time-complexity of $\mathsf{BGRPath}_n$ as

$$\tau(n) \leq 2 \cdot \tau(\mathsf{RBPath}_{n-1}) - 2 + 2 \cdot \tau(n-1) + 2 = 2 \cdot (3^{n-1} + \tau(n-1)),$$

where we used the fact that in Item 6 the first node remains grey-pebbled, which clearly saves at least two steps. For (iii), note that the first node is either not pebbled or black-pebbled in Item 1, not pebbled throughout Items 2 to 4, never pebbled red by induction hypothesis in Item 5, and left grey-pebbled in Item 6. □

Extending to layered graphs. The above strategy can easily be extended to layered graphs of depth D and width w by pebbling the whole layer in which a pebble lies. This will require $O(w \log(D))$ greyscale pebbles. However, since the number of pebbles that this strategy requires is proportional to the width of the circuit (and therefore the input-length), the resulting security bounds in Theorem 22 do not yield any advantage over simply guessing the input. We partially remedy this in the next sections.

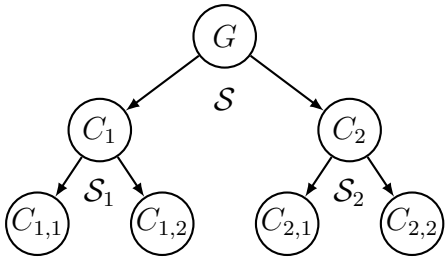


Figure 6.4: Binary component tree of a graph G (for simplicity). The graphs C_1 and C_2 on the second level denote the two components that result from the separator $\mathcal{S}_1$ for G. Similarly, the vertices that form the leaves result from further splitting of C_1 and C_2 using $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively.

6.4.2 BGR Pebbling via Separators

The strategy we describe, BGRSwitch, is implicit in the simulation in [GJ16]. Intuitively, it is obtained by substituting the notion of "midpoint" in BGRPath, the strategy for paths discussed in Section 6.4.1, with that of a graph separator from Definition 49. As a consequence of Theorem 19, a graph G with treewidth $w(G)$ can be recursively decomposed using separators of size at most $w(G)$ into smaller and smaller "component" sub-graphs till the sub-graph is of a manageable (constant) size.[15] As a result, one gets a "component tree" out of the graph, starting with the whole graph at the root and ending with manageable-sized sub-graphs as leaves (see Figure 6.4). For a graph with N vertices and degree δ, the depth of the component tree is at most $O(\log N)$ and its out-degree is at most $\delta \cdot |\mathcal{S}|$ for any separator $\mathcal{S}$ (since each vertex in $\mathcal{S}$ can be connected to at most δ components). The pebbling strategy using separators exploits this recursive structure to minimise the number of greyscale pebbles employed.

Remark 8. Note that Theorem 19 does not provide any guarantees on whether such a sequence of separators can be found *efficiently*. This becomes crucial when simulating the hybrids since it determines the factor t_{P}. We address this question at the end of this section.

RB Pebbling via Separators.

As in the case of BGRPath, we first describe RBTreewidth, a space-efficient RB pebbling strategy that will be used as a subroutine in BGRSwitch. RBTreewidth places a black pebble on any vertex on a graph G of size N and treewidth w using $\sigma := O(w \log(N))$ pebbles. To the best of our knowledge, this strategy is new and might be of independent interest. Since the strategy is reversible, by RBTreewidth^{-1} we denote the reverse strategy that *removes* a black pebble. This strategy, thanks to the observation in Remark 6.2, will also be used to both place or remove a grey pebble on an all-red-pebbled graph (RGRTreewidth).

Lemma 29. *Every node in a DAG G with N vertices and treewidth w can be black-pebbled following the RB pebbling rules using at most $\sigma := O((\delta_{in} + w)\log(N))$ black pebbles in at most $\tau := (\delta_{in} w)^{O(\log(N))}$ steps.*

Lemma 30. *Starting from the all-red configuration, every node in a DAG G with N vertices and treewidth w can be grey-pebbled following the BGR pebbling rules using at most $\sigma := O((\delta_{in} + w)\log(N))$ grey(scale) pebbles in at most $\tau := (\delta_{in} w)^{O(\log(N))}$ steps.*

[15] This exploits the fact that treewidth is a monotone property: removing vertices or edges from a graph can only decrease its treewidth.

Proof of Lemma 29. We denote the pebbling strategy by RBTreewidth and it takes as input a graph (component) C and a vertex v^* to be pebbled. It uses the same recursive decomposition into components as will be in BGRSwitch (i.e., the component tree). The base case is when the graph C is of small enough size (i.e., with $O(1)$ vertices) and here RBTreewidth simply places a black pebble on v^* using as many black pebbles as needed; i.e.:

1. place black pebbles on all vertices in C in topological order; and then
2. remove the black pebbles on the ancestors of v^* in reverse topological order.

Otherwise, RBTreewidth splits C into smaller components using its separator, recursively places a black pebble on every vertex in the separator in *topological order*, places a black pebble on v^* by recursing on the component C^* that contains v^*. Finally, it recursively removes the black pebbles on the separator in *reverse* topological order. The details are given below.

1. Decomposes C into its components $C_1, \ldots, C_p$ using its separator $\mathcal{S} \subseteq \mathcal{C}$, where $\mathcal{C} = \mathcal{S} \cup \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_p$ and $C_i := C_{|\mathcal{C}_i}$.
2. Recursively place black pebbles on the vertices in $\mathcal{S}$ in topological order (analogous to Item 1). That is, for each vertex $s \in \mathcal{S}$ chosen in topological order:
   a) recursively place a black pebble on each predecessor of s (unless it already carries a black pebble) in topological order,
   b) place a black pebble on s, and
   c) recursively remove the black pebbles on each predecessor of s which is not in $\mathcal{S}$ in reverse topological order.
3. Recursively pebble the component $C^* \in C_1, \ldots, C_p$ which contains v^* (analogous to Item 2).
4. Undo Item 2 by recursively removing the black pebbles on the separator in reverse topological order (analogous to Item 3). That is, for each vertex $s \in \mathcal{S}$ chosen in reverse topological order:
   a) recursively place a black pebble on each predecessor of s in topological order,
   b) remove the black pebble on s, and
   c) recursively remove the black pebbles on each predecessors of s in reverse topological order.

As we explain next, carrying out Items 2 and 4 in topological and reverse topological order, respectively, is crucial for the efficiency (and correctness) of RBTreewidth. Recall that the property of the separator $\mathcal{S}$ guarantees that the components $C_1, \ldots, C_p$ are themselves of small enough size (see Definition 49). Therefore, once $\mathcal{S}$ is pebbled black, RBTreewidth can be called on all the resulting components as there are no edges between the components. However, pebbling a vertex s in $\mathcal{S}$ itself is tricky: the predecessors of s could very well be in different components (since there are no guarantees for the vertices in the separator). However, we do have the guarantee that all the predecessors of a predecessor of s which does not lie in $\mathcal{S}$ belong to the same component or the separator, and are reachable via either source vertices *or* vertices belonging to $\mathcal{S}$. Therefore, as long as the vertices in $\mathcal{S}$ are black-pebbled

in topological order, $\mathcal{S}$ can be completely pebbled in Item 2 by recursing on small enough components. A similar argument applies when the black pebbles on the separator are removed in Item 4.

With this in mind, we now analyze RBTreewidth. The reason this strategy requires at most $\sigma := O(w \log(N))$ black pebbles is similar to what we will see in the proof of Theorem 23. The number of pebbles is governed by the expression

$$\sigma(i) \leq (w + \delta_{in}) + \sigma(i+1), \tag{6.3}$$

where the index i is the depth of the recursion of RBTreewidth. The factor $(w + \delta_{in})$ is the cost of black pebbles placed on the separator in Item 2 and the factor $\sigma(i+1)$ is the cost of recursions in Items 2 to 4. Note that since the size of the components in each of these recursive calls is at most $2/3$ of the size of the original component C, the overall depth of the recursion remains $O(\log(N))$. The upper bound on the number of pebbles claimed in the lemma follows by solving Equation (6.3).

As for the number of steps, it is governed by the expression

$$\tau(i) \leq \tau(i+1)O(\delta_{in} w). \tag{6.4}$$

since RBTreewidth is recursively called at most $O(\delta_{in} w)$ times on the (sub-)components. As in the case of space-complexity, since we end up with constant-size components at the end of the recursion, the base cost is $O(1)$. The lemma follows by solving Equation (6.4). □

Recursive Switching.

We are now primed to describe BGRSwitch. It takes as input:

1. the original graph $G = (\mathcal{V}, \mathcal{E})$ that is to be pebbled,
2. the vertices $\mathcal{C} \subseteq \mathcal{V}$ that define the graph component $C = G_{|\mathcal{C}}$ being currently considered,
3. the "higher" separator $\mathcal{U}$, which is the union of all the separators in the "higher" recursive calls that resulted in the creation of the current component C.

Note that $\mathcal{C}$ and $\mathcal{U}$ are disjoint sets by definition. Throughout the execution of BGRSwitch, we maintain a few pebbling properties as invariants:

- At the start of the execution of BGRSwitch on the current component C, it is guaranteed that the vertices in $\mathcal{U}$ are all black-pebbled. This, in some sense, "isolates" C from the rest of the graph and, as a result, it can be pebbled independently of the rest.
- At the end of the execution of BGRSwitch, we guarantee that the vertices $\mathcal{C}$ in C are red-pebbled (via black and then grey), *except* for the children of the higher separator $\mathcal{U}$, which will be left grey-pebbled.

Next, let's see what happens in BGRSwitch when called on $(G, \mathcal{C}, \mathcal{U})$ (in the first call $\mathcal{C} = \mathcal{V}$ and $\mathcal{U} = \emptyset$). (For ease of understanding, in parenthesis we refer to the analogous step in BGRPath, the BGR pebbling strategy for paths that was described in Section 6.4.1.) The base case is when the current component $C := G_{|\mathcal{C}}$ is of small enough size (i.e., with $O(1)$ vertices). Here BGRSwitch simply switches C to red by using as many pebbles as needed; i.e.:

1. place black pebbles on all vertices in C (Rule 50.1),

2. replace them with grey pebbles (Rule 50.3), and
3. replace the grey pebbles with red pebbles (Rule 50.5) *except* if the vertex is a child of the upper separator $\mathcal{U}$.

Otherwise, BGRSwitch does the switching from no pebbles to red pebbles for C by recursively splitting into smaller components using the separator for C as follows (see Figure 6.5).

1. Decompose $C = G_{|\mathcal{C}}$ into its components $C_1, \ldots, C_p$ using its separator $\mathcal{S} \subseteq \mathcal{C}$, where $\mathcal{C} = \mathcal{S} \cup \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_p$ and $C_i := G_{|\mathcal{C}_i}$. (Midpoint-separator analogy)
2. Place black pebbles on the vertices in $\mathcal{S}$ using RBTreewidth. Note that this is possible only because all the vertices that are required to carry this out are either empty or belong to $\mathcal{U}$ and therefore are black-pebbled. (Analogous to Item 1)
3. Recursively switch each component $C_1, \ldots, C_p$ using BGRSwitch. After all component are switched, all vertices in $\mathcal{C}$, except the ones that are children of $\mathcal{S}$, are red-pebbled; the children of $\mathcal{S}$ are left grey-pebbled. (Analogous to Items 2 and 5)
4. Replace the black pebbles on the separator $\mathcal{S}$ with grey pebbles by using Rule 50.3.
5. Replace the grey pebbles on $\mathcal{S}$ and its adjacent vertices with red pebbles (except if the vertex is a child of the upper separator) using Rule 50.5 and RGRTreewidth. (Analogous to Item 6)

Note that during the whole strategy, we maintain as invariant a black-grey frontier between the empty and red-pebbled vertices, and this frontier is exactly at the separators. That is, at any point of the pebbling *no* two vertices such that one is empty and the other is red-pebbled are related (see Remark 6.5). As pointed out in the technical overview (Section 6.1.2), it is this frontier that insulates the computations in the two circuits and help ensure correctness at all times. In the following theorem we formally analyze its space- and time-complexity.

Theorem 23 (Main theorem)**.** *Every DAG G with N vertices, degree δ and treewidth w can be BGR-pebbled using at most $\sigma = O((\delta_{in} + w\delta_{out}) \log(N))$ greyscale pebbles in at most $\tau := (\delta w)^{O(\log(N))}$ steps.*

Proof. To bound the space-complexity of BGRSwitch, first note that the algorithm indeed maintains the invariants stated above: 1) At the start of the execution of BGRSwitch on the current component C, it is guaranteed that the vertices in $\mathcal{U}$ are all black-pebbled. Hence, on input a component at depth $i \in [0, O(\log(N))]$, there are $|\mathcal{U}(i)| = O(iw)$ pebbles that remain black-pebbled. 2) At the end of the execution of BGRSwitch, all the vertices $\mathcal{C}$ in C are red-pebbled, *except* for children of the higher separator $\mathcal{U}$, which will be left grey-pebbled. Hence, after the execution of BGRSwitch on a component at depth $i > 0$ there are up to $\delta_{out} \cdot |\mathcal{U}(i)| = O(i\delta_{out}w)$ many grey pebbles on the graph.

Now, in Item 3 there are up to δw many components, among which some are already switched, some not, and one is currently processed. For the former set of components, all nodes within these which are children of $\mathcal{U} \cup \mathcal{S}$ are pebbled grey. Hence, by the above, there are up to $\delta_{out} \cdot (|\mathcal{U}(i)| + 1) + |\mathcal{U}(i)| = O(i\delta_{out}w)$ nodes that remain grey- or black-pebbled while BGRSwitch is processed on a lower component. Now, while some node on the separator $\mathcal{S}'$ in the currently processed component is pebbled using RBTreewidth or

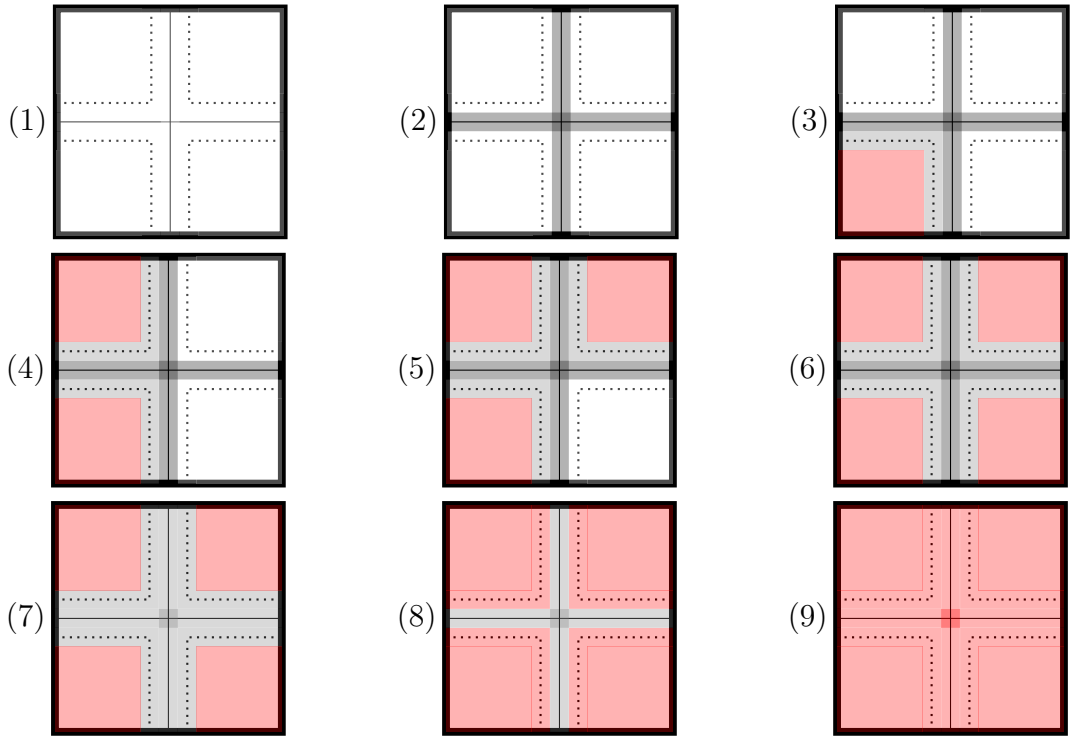


Figure 6.5: A schematic diagram demonstrating recursive switching using BGRSwitch. The separator $\mathcal{S}$, shown as a solid cross, divides the graph (in thick) into four components. The dotted lines indicate vertices adjacent to $\mathcal{S}$. The switching starts from an empty graph (1) and then proceeds as follows: (2) pebble $\mathcal{S}$ black, (3-6) switch bottom-left, top-left, top-right and bottom-right components (in that order), (7) replace black pebbles on $\mathcal{S}$ with grey pebbles, (8) replace the grey pebbles on vertices adjacent to $\mathcal{S}$ with red pebbles, and (9) replace the grey pebble on $\mathcal{S}$ with red pebbles.

RGRTreewidth (cf. Item 2), there are up to $|\mathcal{S}'| \leq w$ additional nodes that remain pebbled. By Lemmas 29 and 30, the space-complexity of RBTreewidth/RGRTreewidth is bounded by $O((\delta_{in} + w)\log(N))$. Thus, we arrive at

$$\sigma(i) \leq O(i\delta_{out}w) + w + O((\delta_{in} + w\delta_{out})\log(N)) = O(w\delta\log(N)).$$

As for the number of steps, on input a component at depth i, the time-complexity of BGRSwitch is governed by the expression

$$\tau(i) \leq (\delta_{in}w)^{O(\log(N))} \cdot w + \tau(i+1)\delta w. \tag{6.5}$$

The first factor is the cost of the subroutines used to pebble the separator black: the subroutine is called at most $|\mathcal{S}| \leq w$ times, each time incurring a cost of at most $(\delta_{in}w)^{O(\log(N))}$ (Lemma 29). The second component is the cost of recursively calling BGRSwitch on at most δw (sub-)components. Since we end up with constant-size components at the end of the recursion, the base cost is $O(1)$. On solving Equation (6.5), the theorem follows. □

Computing separators. Finally, let us get back to the question of how to compute the sequences of separators underlying our pebbling strategy. While we are not aware of an efficient algorithm computing balanced separators, for our purposes (since we lose a similar factor in

the distinguishing advantage) it is enough to note that a separator of size w can be found by brute-force search in time at most N^w by simply enumerating all w-sized subsets of vertices (note that given a separator it is easy to verify that it is indeed one). Since computing any BGR pebbling configuration requires knowledge of at most $O(\log(N))$ many separators, the total time required to compute a pebbling configuration is at most $t_{\mathsf{P}} = O(\log(N)N^w)$.

6.4.3 Optimised Piecewise Guessing

Recall that in Theorem 22, the loss in adaptive security is exponential in the BGR pebbling complexity. This is because the reduction requires as advice the value of the output wire of all the gates that are greyscale pebbled. Therefore, when Theorem 22 is used in conjunction with Theorem 23, the loss is exponential in the treewidth *as well as* degree. First, note that for Yao's garbling scheme, we only consider Boolean circuits with fan-in 2. We argue next that the dependence on the degree can be removed thanks to the structure of the configurations in the BGRSwitch pebbling strategy. The resulting theorem is stated in Theorem 24.

Let's return to the recursive step in Item 3 which is the cause of the dependence on the degree. At the start of this step, all the vertices in the separator $\mathcal{S}$ have been pebbled. Then each component C_i is recursively switched to red one at a time. At the end of switching C_i, each vertex in C_i is pebbled red, *except* for those vertices that are children of $\mathcal{S}$ (or $\mathcal{U}$) which are left grey. Therefore we can restrict our focus on those vertices that have its predecessors in the separator – let's consider one such vertex v^*. Note that in any configuration where v^* carries a grey pebble, it is guaranteed that its predecessors in the separator are black-pebbled. Therefore, instead of requiring the value of the gate g^* corresponding to v^* as an advice, it can simply be computed as a function of the values of its predecessor gates (which *are* included in the advice). To sum up, instead of providing as advice the values of the output wires of *all* the gates that are greyscale pebbled as in Equation (6.2), it suffices to provide a much smaller advice as outlined above. As a result of this observation, we get the following optimised version of Theorem 22. This leads to the corollaries stated in Section 6.1.1.

Theorem 24. *Suppose that a class of circuits $\mathcal{C}$ of size N, fan-in 2 has degree δ and treewidth w. If the encryption scheme* SKE *is (t, ϵ)-secure then* $\mathsf{YGC}_{\mathsf{SKE}}$ *is $(t - (t_{\mathsf{H}} + t_\sigma + t_{\mathsf{P}}), 3\tau 2^\sigma \cdot \epsilon)$-adaptively-indistinguishable for $\mathcal{C}$ where*

$$\tau := (\delta w)^{O(\log(N))}, \quad \sigma := O(w \log(N)) \quad \textit{and} \quad t_{\mathsf{P}} := O(\log(N)N^w).$$

6.5 Conclusion and Open Problems

Yao's garbling scheme is one of the most fundamental cryptographic constructions. In this work, we took another step towards completing the landscape of its security. Our result leads to several interesting questions, the most natural being whether the upper bound on loss in security can be improved. To this end, one could look at other (orthogonal) graph properties. Another pressing question is whether there are other applications of treewidth in cryptography (which seems relatively overlooked compared to other fields such as circuit complexity or algorithmic graph theory). This closely concerns the divide-and-conquer approach employed in our security reduction: it seems that the approach of surgically replacing one circuit with another should find use in other scenarios. Our hope is that this work spurs further research in this direction.

Part IV

Lower Bounds

CHAPTER 7

On the Cost of Adaptivity in Security Games on Graphs

7.1 Introduction

In the previous chapters we have seen several cryptographic schemes, for which adaptive security seemed hard to achieve: Generalized selective decryption (Section 3.3), the GGM construction of a prefix-constrained psudorandom function (Section 3.4), proxy re-encryption schemes (Chapter 4), the TreeKEM protocol for continuous group key agreement (Chapter 5), and Yao's garbling scheme (Chapter 6). For all these schemes, the best known security reductions (in the standard model) applied the Piecewise-Guessing framework that we presented in Chapter 3, and the technical difficulty consisted in finding good pebbling strategies for various pebbling games on some underlying graph structures. It is a natural question whether the existing security proofs obtained in the Piecewise-Guessing framework can be further improved. In this chapter, we approach this question from the negative direction and argue that simply using existing techniques, this will not be possible for some of the applications.

7.1.1 Our Results

We show that there do not exist non-rewinding PPT black-box reductions – henceforth called "straight-line" reductions for brevity – that prove adaptive security of

- certain forms of restricted GSD (including its public key variant, see Section 3.3) based on the IND-CPA security of the underlying encryption scheme (Section 7.6),
- popular protocols for continuous group key agreement (CGKA, see Chapter 5) based on the IND-CPA security of the underlying public-key encryption (PKE) scheme (Section 7.7),
- the GGM construction for prefix-constrained PRFs (see Section 3.4) based on the pseudorandomness of the underlying PRG (Section 7.8)

This Chapter essentially replicates, with permission, the full version [KKPW21b] of our publication [KKPW21a], © IACR 2021, to appear.

Application	Underlying Graph	Lower Bound	Reduction	Upper Bound
GSD	Path P_N	$N^{\Omega(\log(N))}$	Oblivious	$N^{\mathsf{O}(\log(N))}$[FJP15], §3.3
	Binary In-Tree B_n	$N^{\Omega(\log(N))}$	Oblivious	$N^{\mathsf{O}(\log(N))}$ [Pan07], §3.3
	Tree[1]	$N^{\Omega(\log(N))}$	Straight-line	$N^{\mathsf{O}(\log(N))}$[FJP15], §3.3
	Arbitrary DAG	$2^{\Omega(\sqrt{N})}$	Oblivious	$N^{\mathsf{O}(N)}$ §3.3
PRE	Path P_N	$N^{\Omega(\log(N))}$	Oblivious	$N^{\mathsf{O}(\log(N))}$ §4
	Binary Tree B_n	$N^{\Omega(\log(N))}$	Oblivious	$N^{\mathsf{O}(\log(N))}$ §4
	Arbitrary DAG	$2^{\Omega(N)}$	Arbitrary	$N^{\mathsf{O}(N/\log(N))}$ §4
GGM cPRF	Tree	$n^{\Omega(\log(n))}$	Straight-line	$n^{\mathsf{O}(\log(n))}$ [FKPR14], §3.4
TreeKEM	Regular Tree	$M^{\Omega(\log(\log(M)))}$	Straight-line	$Q^{\mathsf{O}(\log(M))}$ §5

Table 7.1: Summary of lower bounds on the loss in security established in our work. $N = 2^n$ denotes the size of the graph. Therefore, in the case of GGM constrained PRF, n denotes the length of the input string. For TreeKEM, M denotes the number of users and Q refers to the number of queries allowed to the adversary.

- proxy re-encryption (PRE) schemes (see Chapter 4) based on the IND-CPA security of the PKE scheme and N-weak key privacy (Section 7.9)

with only polynomial loss in advantage. For PRE we can even rule out general (i.e., rewinding) black-box reductions (see Discussion of Corollary 18). For the theorem statements of the latter three results, we refer to the corresponding sections, but we will discuss GSD in a little more detail, so we provide an informal statement here.

Informal Statement 1 (Corollary 9)**.** *Any straight-line reduction proving security of unrestricted adaptive* GSD *based on the IND-CPA security of the underlying symmetric-key encryption scheme loses at least a factor that is* super-polynomial *($N^{\Omega(\log N)}$) in the number of users N.*

For the proof we rely heavily on the adversary's freedom to query arbitrary directed acyclic graphs (DAG). (Actually, the graphs have some structure and so certain conditions may be imposed on it but these restrictions are very weak.) In many applications however, the adversary is much more restricted in terms of the graphs it can query, e.g. in protocols for multicast encryption like logical key hierarchies (LKH) [WHA98, WGL00, CGI+99], and hence our bound does not apply. However, for a certain sub-class of straight-line reductions, which we term "oblivious" (see discussion below), we obtain results for such applications. These results show that the upper bounds for GSD given in 3, which are oblivious, are essentially tight and can only be improved by exploiting new non-oblivious techniques (and similarly for the bounds for PRE given in 4), as stated informally below.

Informal Statement 2 (Corollaries 6 to 8)**.** *Any oblivious reduction proving security of adaptive GSD restricted to* paths or binary trees *based on the IND-CPA security of the underlying symmetric-key encryption scheme loses a factor that is* super-polynomial *($N^{\Omega(\log N)}$) in the number of users N; for unrestricted GSD the loss is* sub-exponential *($2^{\Omega(\sqrt{N})}$).*

Our results for PRE have a similar flavor, but are even stronger, since in this case the reduction is naturally more restricted. A summary of the results can be found in Table 7.1.

[1]Recall that a tree does not necessarily have to be rooted, so this includes any DAG such that the corresponding undirected graph does not contain any cycles.

The common thread to the applications we consider is that their security game can be abstracted out by a two-player multi-stage game which we call the "Builder-Pebbler Game". We are unable to establish lower bounds for other applications of the Piecewise-Guessing framework (e.g., computational secret sharing or garbling circuits) from the results in this chapter, because their security model is not quite captured by the Builder-Pebbler Game. The high level reason for this is that the graphs (e.g., the circuit to be garbled or the access structure) in these applications is fixed ahead of time and the adaptivity comes from other sources (e.g., choice of garbling input or targeted user). Therefore we would require other combinatorial abstractions to establish lower bounds for them. In fact, building on the high level ideas introduced in this chapter, in Chapter 8 we show lower bounds for adaptive security of Yao's garbling (see Section 8.2.5 for a comparison).

We defer the discussion on the Builder-Pebbler Game to the next section (Section 7.2.2) and explain informally what we mean by oblivious reductions next, mostly from the perspective of GSD. We will then argue that this comprises a natural class of reductions.

Oblivious reductions. Oblivious reductions are a certain class of black-box reductions and our definition is motivated by the reductions in 3 and other applications of the Piecewise-Guessing framework. On a high level, the behaviour of an oblivious reduction is "independent" of the adversary's behaviour throughout the simulation of the security game. To see what we mean by this, let's return to the example of GSD. A reduction (simulating some consecutive hybrids) can decide to answer an encryption query issued by the adversary either with a consistent or an inconsistent ciphertext (let's ignore the challenge ciphertext for the moment). In particular, it has total control over the number of inconsistencies in the final simulation (assuming it knows the number of queries the adversary will make). However, as the key-graph is only gradually revealed to the reduction, it doesn't know where the edge (representing the encryption query) will end up within the key-graph. We call a GSD reduction *oblivious* if it does not make use of the partial graph structure it learns during the game but rather sticks to some strategy that is independent of the history of the adversary's queries. There are several ways one could formalise this: for example, one could require the reduction as initially "committing" to which queries it will answer inconsistently. However, this does not mean that for all queries it has to commit to its decision, but rather commit to some minimal description of the edges it intends to respond inconsistently to. In order to capture as many reductions as possible (while still being able to prove lower bounds), we ended up defining them as reductions which commit to a minimal set of nodes which *covers* all inconsistent edges, i.e., a minimal *vertex cover*.[2] For example in the case of graphs of high indegree, clearly, guessing the set of sinks of inconsistent edges gives a much more succinct representation. A formal definition of an oblivious GSD reduction is given in Definition 68; the corresponding definition for PREs is given in Definition 71.

Why oblivious reductions? We note that oblivious black-box reductions are a quite natural notion, since they can easily be defined uniformly for all adversaries. Not surprisingly, they encompass some of the key reductions in the literature. Beside the reductions proposed and analyzed in this work (and other concurrent works applying the Piecewise-Guessing framework), partitioning-based reductions, which have been successfully employed in a plethora of works

[2]Technically, we do not require *minimal* vertex cover, but a weaker notion which we call "non-trivial" vertex cover (see Definition 52).

[Cor00], also roughly behave in an oblivious manner.[3] Moreover, oblivious black-box reductions encompass the currently-known techniques for establishing upper bounds for primitives with dynamic graph-based security games, like GSD, PRE, cPRFs etc. Therefore, our results imply that in order to obtain better upper bounds on the loss function Λ even in the more restricted settings, one needs to deviate significantly from the current proof techniques (i.e. non-oblivious or rewinding reductions for GSD and restricted PRE). Accordingly, our results on oblivious reductions should not be viewed as separations, but rather as a guide towards new avenues to finding better reductions by ruling out a large class of reductions – such possibilities are discussed in Section 7.10.

7.1.2 Related Work

Black-Box Separations

The study of limitations of black-box reductions was initiated in the seminal work of Impagliazzo and Rudich [IR89]. They used the notion of *oracle separations* to rule out black-box reductions of key agreement to symmetric-key primitives. This approach turned out quite useful and has been further exploited to rule out black-box constructions of a variety of cryptographic primitives from one another (e.g., [Rud88, Sim98]). A fine-grained study of the notion of black-box reductions and oracle separations was later carried out by Reingold et al. [RTV04].

In addition to ruling out reductions, oracle techniques have also been used to study the efficiency of a construction of one primitive from another [GT00, GGKT05, KST99]. This has been applied to the case of adaptive security as well. Perhaps the works most relevant to ours is that of Lewko and Waters [LW14], who showed that the security of adaptively-secure hierarchical identity-based encryption must degrade exponentially in the depth, and Fuchsbauer et al. [FKPR14], who showed that certain types of constrained PRFs must incur an exponential loss (in the size of the input) in adaptive security. Note that this class of constrained PRFs does not include the prefix-constrained PRF construction we consider in this work. Both aforementioned works employ the more recent meta-reduction technique [BV98, GW11, Pas13], which is of different flavour from oracle separations. Our work is thus similar in spirit to [GT00, GGKT05, KST99].

Graph Pebbling

The notion of graph pebbing, first introduced in the 70's to study programming languages, turned out quite useful in computational complexity theory to study the relationship between space and time; in recent years, pebbling has found applications in cryptography as well [DNW05, DKW11b, AS15]. The notion of node pebbling first appeared (albeit implicitly) in [PH70], whereas the notion of *reversible* node pebbling was introduced by Bennett to study reversible computation [Ben89]. The notion of edge pebbling used in this work is defined in 3 . The lower bound on the reversible node pebbling complexity of paths was established by Chung et al. [CDG01] and an alternative proof can be found in [Krá01]. As for the lower bound on the node pebbling complexity for binary trees, a proof can be found in [Sav98]. We refer the reader to the textbook by Savage [Sav98] or the excellent survey by Nordström [Nor15] for more details on pebbling.

[3] On every signature query issued by the adversary, the reduction in [Cor00] tosses a (biased) random coin (*independent* of the history of the simulation) and depending on its outcome decides whether or not to embed the (RSA) challenge in the signature. The simulation is identical if these coin-tosses are all carried out together at the beginning of the game.

7.2 Technical Overview

On a high level, our approach can be divided into two steps. In the first step (Section 7.2.2), which is purely combinatorial, we analyze a two-player multi-stage game which we call the Builder-Pebbler Game. In particular, we exploit ideas from pebbling lower bounds to establish upper bounds for the success probability of the Pebbler (who is one of two players). These upper bounds are then, in the second step (Section 7.2.3), translated to lower bounds on the loss in security of concrete cryptographic protocols using oracle separation techniques to yield the results stated in Section 7.1.1. Before explaining the two steps, we provide a summary of the overall approach so that the two steps, especially the motivation behind some of the underlying definitions, can be better appreciated.

7.2.1 Our Approach

Our goal is to design adversaries that break the GSD game but where any reduction (in a specified class) to the security of the underlying SKE scheme loses a significant (super-polynomial) factor in the advantage. Since we are aiming to rule out black-box reductions, we have the luxury of constructing inefficient adversaries and SKE schemes. The output of our adversaries will solely depend on the distribution of inconsistent edges in the final key-graph, which we will denote as *pebbles* in the following. Clearly, in order to win the GSD game, our adversaries need to output 0 if the final key-graph is entirely consistent (i.e., contains no pebbles), and 1 if the final key-graph is entirely consistent except for the edges incident on the challenge key. Otherwise, we have complete freedom in assigning output probabilities of 0 and 1 to the remaining pebbling configurations of the final key-graph.

As we prove formally in Section 3.3, any reduction attempting to take advantage of our adversaries must send its IND-CPA challenge as a response to a query and exploit the fact that the real and the random challenge will lead to different pebbling configurations of the key-graph. Its hope is that the output distribution of the adversary differs significantly between the two configurations. Note however, that when embedding the challenge in some edge (i, j) of the key-graph, all edges incident to i will, with overwhelming probability, be inconsistent independently of the challenge ciphertext, since the reduction does not know the challenge secret key and thus is unlikely to be able to send consistent responses to queries incident to i. In other words, the challenge can only be embedded into an edge where the edges incident to the source are all pebbled. This naturally leads to studying configurations that are related by valid moves in the *reversible edge-pebbling* game: a pebble on an edge may only be added or removed if all edges incident to the source are pebbled.

We may now define the configuration graph of our key-graph G: The vertices of the configuration graph $\mathcal{P}^G$, as the name suggests, consist of all possible pebbling configurations of G. Therefore it is the power set of the edges of $G = (\mathcal{V}, \mathcal{E})$. An edge is present from a vertex $\mathcal{P}_i$ to another vertex $\mathcal{P}_j$ if $\mathcal{P}_j$ can be obtained from $\mathcal{P}_i$ using a valid pebbling move. The edges represent pairs of configurations, where the reduction may embed its IND-CPA challenge, in other words, a hybrid (from the reduction's point of view). Since we consider reversible pebbling games, the edges in our configuration graphs are undirected. Therefore one can think of $\mathcal{P}^G$ as a subgraph of the Boolean hypercube on $2^{|\mathcal{E}|}$ vertices. Assuming that G has a single sink vertex T, $\mathcal{P}^G$ has two special vertices denoted $\mathcal{P}_{\text{start}} = \emptyset$ and $\mathcal{P}_{\text{target}}$ which consist of the pebbling configuration where all incoming edges to T carry a pebble. The configuration graph for C_4, the path of length 4, is given in Figure 7.1. A more formal definition is given later in Definition 55 (Section 7.3). A path from $\mathcal{P}_{\text{start}}$ to $\mathcal{P}_{\text{target}}$ corresponds to a pebbling sequence

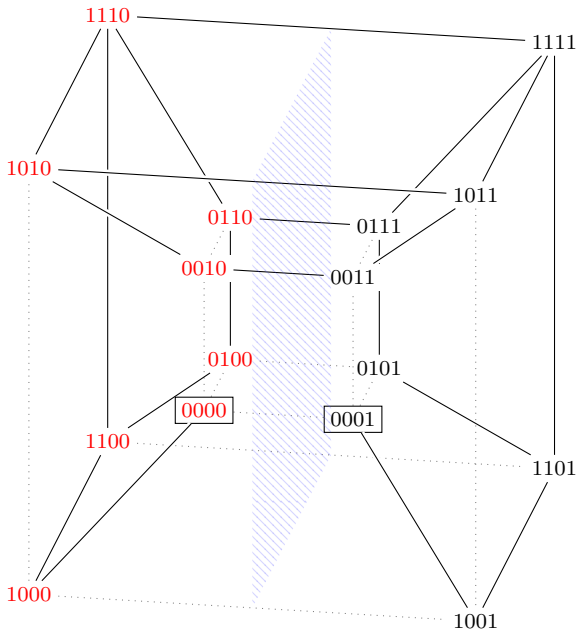


Figure 7.1: Configuration graph for paths of length 4, $C_4 = ([1,5], \{(1,2),(2,3),(3,4),(4,5)\})$. It is a subgraph of the Boolean hypercube of dimension four (the missing edges are dotted). The labels of the vertices encode the pebbling status of the corresponding edge and therefore represent a pebbling configuration: e.g., the vertex labelled 0000 is completely unpebbled (configuration $\mathcal{P} = \emptyset$) whereas the vertex labelled 1000 has a pebble only on the first edge $(1,2)$ (configuration $\mathcal{P} = \{(1,2)\}$). An edge exists between a configuration $\mathcal{P}_i$ and $\mathcal{P}_j$ if $\mathcal{P}_j$ can be obtained from $\mathcal{P}_i$ via *one* valid pebbling move. The special vertices for $\mathcal{P}^{C_4}$ are $\mathcal{P}_{\text{start}} = 0000$ and $\mathcal{P}_{\text{target}} = 0001$ (both boxed). A cut for this configuration graph is indicated by the hatched plane (which lies parallel to the $1000 - 1010 - 1110 - 1100$ and $1001 - 1011 - 1111 - 1101$ planes). The cut consists of the set of (red) vertices that lie on the left of the plane, i.e., the set of vertices with the least significant bit 0: $\{0000, 0010, 0110, 0100, 1000, 1010, 1110, 1100\}$. The subset of edges that intersect this plane form the cut set, i.e., the edges which flip the least significant bit: $\{(1110,1111),(1010,1011),(0010,0011),(0110,0111)\}$. This cut is of a geometric nature – the cuts that we deploy in our lower bound, on the other hand, will have a more combinatorial underpinning (see Sections 7.2.2 and 7.5).

in the reversible edge-pebbling game. Any such path can be used for a hybrid argument to prove upper bounds for the loss in security, which is what prior works did [Pan07], Section 3.3. In this work we are interested in ruling out the possibility of using any of the paths (or multiple at once) to improve on these results.

Pebbling lower bounds: Barriers to better cryptographic upper bounds. In our approach, we will show that in *any* sequence of hybrids there exist "bottleneck" configurations related to pebbling lower bounds. These bottleneck configurations define a cut for the configuration graph $\mathcal{P}^G$. Looking ahead, our adversaries will concentrate all their advantage on these cut sets and we will show that it is hard for any reduction to guess the pebbled edges of the corresponding pebbling configurations.

For example, let's consider the pebbling lower bound for binary trees. It is known that the number of pebbles that are needed to *node*-pebble a complete binary tree of N vertices is at least $\log(N)$ (see [Sav98] for example), and the argument can be easily adapted for the case of edge pebbling as follows. Consider a pebbling sequence for a complete binary tree. At the

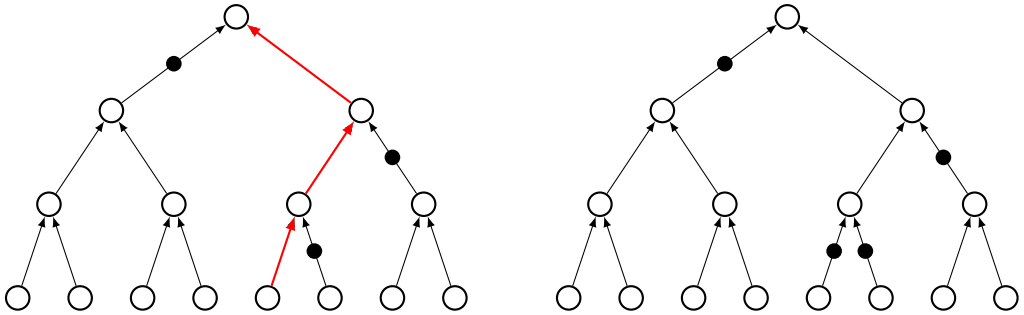


Figure 7.2: The pebbling configurations used to argue lower bounds for edge-pebbling of a perfect binary tree B_3 of depth 3. In the left configuration there exists a path from a leaf (100) to the root (ε) that is *not* covered by a pebble highlighted by the thicker (red) path. In the right configuration *all* the paths in B_3 are covered by pebbles. The cut is defined at such two configurations.

beginning of the sequence *none* of the $N/2$ paths from the leaves to the root carries a pebble; whereas at the end of the sequence – at which point both the edges incident on the root must carry a pebble – *all* the paths from the root to the leaves carry a pebble. Furthermore, by the rules of edge pebbling, only new pebbles on edges going out of the sources, i.e., the leaves, increase the number of paths that carry a pebble. So any pebbling move can only decrease the number of paths that carry a pebble by one. Therefore there have to exist two consecutive configurations in the pebbling sequence such that in the first configuration there exists a path that does not carry a pebble but in the next configuration every path carries a pebble (see Figure 7.2). At this point, for each node on the path (except the leaf) there must be at least one pebble on the graph to pebble all paths going through this node via the other in-going edge, and therefore there exists a pebbling configuration where there are at least $\log(N)$ pebbles. Such pairs of configurations will serve as the cut for the case of binary trees. An illustration can be found in Figure 7.2.

From pebbling lower bounds to cryptographic lower bounds via Builder-Pebbler Game. The immediate idea would be to translate pebbling lower bounds directly to cryptographic lower bounds. But pebbling lower bounds apply to fixed graphs. Therefore we are missing a component that captures the dynamic nature of the security games, like that of GSD, which involves (the adversary) choosing a graph G randomly from a class of graphs $\mathcal{G}$. To remedy this, we introduce a two-player multi-stage game that we call the Builder-Pebbler Game and then show that pebbling lower bounds can be used to upper bound the probability of success of the Pebbler (Step I: Section 7.2.2), one of the players. Then we will use oracle separation techniques to translate these upper bounds into cryptographic lower bounds (Step II: Section 7.2.3).

7.2.2 Step I: Combinatorial Upper Bounds

We start off with an informal description of the Builder-Pebbler Game, a two-player game that will abstract out the combinatorial aspect of establishing lower bounds for cryptographic protocols that are modelled by multi-user games where the adversary adaptively builds a graph structure among the set of users, as in GSD (formal definition in Section 7.4). The game is played between a Pebbler and a Builder, and intuitively, Pebblers play the role of reduction algorithms whereas Builders correspond to adversaries in security games.

Builder-Pebbler Game. For a parameter $N \in \mathbb{N}$, the Builder-Pebbler Game is played between a Builder and a Pebbler in rounds. The game starts with an empty DAG $G = (\mathcal{V} = [1, N], \mathcal{E} = \emptyset)$ and an empty pebbling configuration $\mathcal{P}$, and in each round the following happens: the Builder first picks an edge $e \in [1, N]^2 \setminus \mathcal{E}$ and adds it to the DAG and the Pebbler then decides whether or not to place a pebble on e. This way the Builder and the Pebbler will construct a graph G and a pebbling configuration $\mathcal{P}$ on this graph. The Builder can stop the game at any point by choosing a sink in G as the challenge. This results in a *challenge* DAG $G^* = (\mathcal{V}^*, \mathcal{E}^*)$, the subgraph of G that is induced by all nodes from which the challenge is reachable. The Pebbler wins if it ends up with a pebbling configuration $\mathcal{P}$ that is in a designated subset of all configurations. This winning set is determined by the graph G. Otherwise, the Builder is declared the winner. In case the strategies are randomised, we call the probability with which the Builder (resp., the Pebbler) wins the game as *Builder's (resp., Pebbler's) advantage*, and denote it by $\beta = \beta(N)$ (resp., $\pi = \pi(N)$). We also consider restricted games where the Builder is restricted to query graphs G that are subgraphs of some family of graphs $\mathcal{G}$.

In summary, one can think of the game as the Builder building a graph and the Pebbler placing pebbles on this graph with the aim of getting into a winning configuration and the Builder preventing this from happening.[4]

Defining winning configurations via cuts of the configuration graph. Although the Builder-Pebbler Game is meaningful for any notion of winning configuration, we are interested in a particular definition that is essential in establishing our cryptographic lower bounds: we will set the winning configurations as the ones that belong to bottleneck configurations in the configuration graph of G. The goal is to prove that it will be difficult for Pebblers to get into such configurations. In some cases we can do so directly, but in others the Pebbler may be able to achieve this by "flooding" the graph with many pebbles. For example, recall the bottleneck configurations of a binary tree, which consists of all configurations where exactly one path from a source to the root is not pebbled. A randomised Pebbler that decides for each query whether to pebble an edge or not using an unbiased coin, will leave any particular path unpebbled with a probability that is inverse polynomial in the number of nodes in the tree. With a bit more effort one can show that such a path is the only unpebbled path with significant probability. Our solution is to "artificially" restrict the Pebbler to placing very few pebbles by requiring it to leave the part of the query graph that is not in the challenge graph entirely unpebbled, i.e. if at the end of the game there is a pebble on an edge that is not rooted in the challenge graph, the Pebbler loses. Note that this does not trivialize our task of finding a suitable Builder, because for our application to cryptographic lower bounds to work, the Builder's querying strategy (including the challenge) needs to be independent of the pebbles placed by the Pebbler. (We call such Builders also *oblivious*, see below.) Returning to our example of binary trees, the Builder may now query two binary trees and select one of them to be the challenge graph at the end of the game. A Pebbler that places too many pebbles will now lose the game with overwhelming probability. Of course, care must be taken that this behaviour cannot be exploited by the reduction. Intuitively, the reason this works

[4] This is reminiscent of Maker-Breaker games [HKSS14], a class of positional games (which includes Shannon Switching Game, Tic-Tac-Toe and Hex) which are played between a Maker, who is trying to end up with a (winning) position and a Breaker, whose goal is to prevent the Maker from getting into such (winning) positions. One fundamental difference between Maker-Breaker Games and the Builder-Pebbler Game is that in Maker-Breaker games one usually considers optimal (deterministic) strategies, whereas we consider randomised strategies for the Builder-Pebbler Game. (Another way of looking at this is that our "board" is dynamic.) Another difference is the asymmetry in the nature of moves.

is that in all our applications, if the reductions were to embed the challenge outside of the challenge graph, our adversaries will almost always interpret it to be a pebble, no matter if the challenge was real or random.

Combinatorial Upper Bounds in the Builder-Pebbler Game. We bound the advantage of Pebblers from above against Builders with varying degree of freedom, i.e., Builders that are restricted to querying certain classes of graphs. The upper bounds in Statements 3 to 5 are (almost) tight since a random Pebbler yields (almost) matching lower bounds.

Informal Statement 3 (Theorems 25 and 27)**.** *Any oblivious*[5] *Pebbler in the Builder-Pebbler Game restricted to* paths or binary trees *has advantage at most inverse* quasi-polynomial *($N^{-\Omega(\log N)}$) in N, the size of the graph.*

Informal Statement 4 (Theorem 28)**.** *Any oblivious Pebbler in the unrestricted Builder-Pebbler Game has advantage at most inverse* sub-exponential *($2^{-\Omega(\sqrt{N})}$) in N, the size of the graph.*

Informal Statement 5 (Theorem 30)**.** *Any Pebbler in the Builder-Pebbler Game restricted to* trees *has advantage at most inverse* quasi-polynomial *($N^{-\Omega(\log N)}$) in N, the size of the graph.*

Remark 9 (On Builder Obliviousness)**.** It is worth mentioning that all our Builder strategies are also *oblivious*, where oblivious is defined different for Builders than for Pebblers: it means that the query strategy is independent of the Pebbler's responses (see Section 7.4.1).[6] The reason we restrict ourselves to such Builders is mostly for our convenience: looking ahead, it means that we can ensure that the reductions in our cryptographic applications cannot exploit the querying behaviour of the adversary to gain a larger advantage, rather they must rely solely on the final output bit.

7.2.3 Step II: From Combinatorial Upper Bounds to Cryptographic Lower Bounds

For translating upper bounds established in Step I into loss in security of concrete cryptographic protocols, we adapt ideas from oracle separations.

Ruling out tight black-box reductions. Oracle separations are used to rule out reductions[7] proving security of a primitive Q (e.g., PKE) based on the security of another primitive P (e.g., SKE). Our case is slightly different since it involves a primitive P (e.g., SKE) that is used in a graph-based "multi-instance" setting Q^P (e.g., GSD with SKE). In this setting, we are interested in the more fine-grained question of bounding Λ, the loss in security incurred by

[5] The notion of obliviousness for Pebblers is naturally derived from the one for reductions, see discussion above and Definition 60.

[6] One could think of the Builder playing the role of "nature" (who also adopts a strategy that is oblivious of the opposing player) in Papadimitrou's *Games Against Nature* [Pap85].

[7] The usage of the word 'reduction' here is in a constructive sense [RTV04]: a primitive Q is reduced to another primitive P if (i) there is an efficient *construction* C that takes an implementation P of P and gives an implementation Q of Q and (ii) there is an efficient *security reduction* R which takes an adversary A_Q that breaks Q and constructs an adversary A_P that breaks P. For example, the most common type of reduction used in cryptography is a *fully* black-box reduction where both R and C are black-box in that they only have black-box access to P and A_Q, respectively.

any efficient black-box reduction R that breaks P when given black-box access to an adversary that breaks Q^P. This means we must show that for *every* R, there exists

- an instance P (not necessarily efficiently-implementable) of P and
- an adversary A_Q (not necessarily efficient) that breaks Q^{P}

such that the loss in security incurred by R in breaking P is at least Λ.[8] To this end, we establish a tight *coupling* between the security game for Q^P and the Builder-Pebbler Game (e.g., Lemma 37). If Q^P involves a graph family $\mathcal{G}$ then the Builder-Pebbler Game will be played on $\mathcal{G}$ (or sometimes another family related to $\mathcal{G}$) and the winning condition is determined by the cut for $\mathcal{G}$. The coupling is established using a Builder strategy B and a related adversary A_Q such that

- every reduction R can be translated to a Pebbler strategy P against B on $\mathcal{G}$, and
- if R has a security loss of at most Λ then B's advantage against P is at least $1/\Lambda$ (up to negligible additive factors).

If $\mathcal{G}$ is a class for which we derived an upper bound of π for Pebbler strategies (in Step I) then any reduction R such that $1/\Lambda > \pi$ cannot exist. Put differently, an upper bound on the success probability of the Pebbler in the Builder-Pebbler Game translates to a lower bound on the loss in security for the reduction R. In the remainder of the section, we explain how the coupling works in a bit more detail using GSD on binary trees as the running example. To keep the exposition simple, we will brush a lot of issues (e.g., dealing with 'flooding' reductions) under the rug and refer the readers to Section 7.6 for a more formal treatment.

Example: GSD on Binary Trees. Let's consider the case where P is SKE and Q^P is the GSD game played on $\mathcal{G} = \mathcal{B}_n$, the class of binary trees of depth n. Intuitively, the GSD adversary A_Q "simulates" the oblivious Builder B used to derive Statement 3. That is, it

1. chooses a binary tree $B_n \in \mathcal{B}_n$ uniformly at random,
2. queries, in a random order, each edge $(u, v) \in E(B_n)$ to obtain the corresponding ciphertext $\mathsf{Enc}_{\mathrm{k}_u}(\mathrm{k}_v)$ from the reduction R and
3. challenges the sole sink T at the end of the game.

For it to be a valid adversary, A_Q must distinguish the extreme games, i.e., the real game where all the ciphertexts are real and the random game where the ciphertexts incoming to T are both random. To this end, it looks at the ciphertexts it obtained and extracts a pebbling configuration $\mathcal{P}$ from it (as described in Section 7.2.1). Note that the extreme hybrids corresponds to $\mathcal{P}_{\text{start}} = \emptyset$ (real) and $\mathcal{P}_{\text{target}}$ such that both the edges incoming to T carry a pebble (random). A_Q distinguishes these by concentrating all its advantage in the cut in the configuration graph of B_n defined in Section 7.2.1: i.e., it outputs 0 if $\mathcal{P}$ is on the "left" of the cut and 1 otherwise (see Figure 7.2). To help A_Q faithfully distinguish real ciphertexts from random ones so that it can infer the exact pebbling configuration $\mathcal{P}$, we fix P to be an ideal implementation $(\mathsf{Enc}, \mathsf{Dec})$ of SKE:

[8] This is obtained by simply negating the definition of a black-box reduction: *there exists an efficient reduction* R *such that for every (not necessarily efficient) implementation* P *of* P *and for every (not necessarily efficient) adversary* A_Q *that breaks* Q^{P} *the loss in security is at most* Λ.

- Enc is a random expanding function that implements encryption and
- Dec is the decryption function defined to be "consistent" with Enc.[9]

Since Enc is injective with overwhelming probability, given a ciphertext A_Q can brute force Enc to determine (exactly) whether or not the ciphertext corresponding to an edge is real. By carrying this out for all the edges, it can extract a *unique* pebbling configuration corresponding to R's simulation. Since A_Q concentrates its advantage in the cut, for R to have any chance of winning, its own challenge c^* must be 'embedded at the cut' so that – depending on whether or not c^* is real – $\mathcal{P}$ switches from left of the cut to its right. Since this is the *only* way R can exploit A_Q, we may infer that a reduction with loss in security at most Λ ends up in the cut with probability at least $1/\Lambda$. However, thanks to the fidelity of the extraction, this also means that the natural Pebbler strategy P that underlies R, which simply places a pebble whenever R fakes, wins against B in the Builder-Pebbler Game on $\mathcal{B}_n$ with an advantage at least $\pi = 1/\Lambda$ (formally, Lemma 34). In particular, if Λ is significantly less than quasi-polynomial in $N = 2^n$, it would imply the existence of a Pebbler that is successful with a probability greater than inverse quasi-polynomial, a contradiction to Statement 3 (formally, Corollary 7). Since Statement 3 only holds for oblivious Pebblers, the bound on Λ only holds for oblivious GSD reductions.

7.3 Notation and Definitions

In this chapter, let $G = (\mathcal{V}, \mathcal{E})$ define a directed acyclic graph (DAG) with vertex set $\mathcal{V} = [1, N]$, edge set $\mathcal{E} \subset [1, N] \times [1, N]$, and a set of sinks $\mathcal{T}$. By B_n, we denote a binary tree of depth n – the binary tree is *perfect* if it has all $2^{n+1} - 1$ vertices. We assume the standard indexing of the vertices in B_n by associating them with binary strings in $\{0,1\}^{\leq n}$ determined by their position in the tree: i.e., the root has index ε and the left (resp., right) child of a vertex with index i is $i\|0$ (resp., $i\|1$). We will consider the edges

Definition 51 (Cuts, cut-sets, frontiers)**.** Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected graph. A *cut* $\mathcal{S}$ of G is a subset of the nodes $\mathcal{V}$. For two nodes $v_1, v_2 \in \mathcal{V}$ an *s-t-cut that separates* v_1 *and* v_2 is a cut $\mathcal{S}$ such that $v_1 \in \mathcal{S}$ and $v_2 \notin \mathcal{S}$. The *cut-set* of a cut $\mathcal{S}$ is the set of edges with one endpoint in $\mathcal{S}$ and the other outside of $\mathcal{S}$. We call the *frontier of a cut* $\mathcal{S}$ the set of all nodes in $\mathcal{S}$ that have an incident edge in the cut-set of $\mathcal{S}$.

Definition 52 (Vertex covers)**.** Let $G = (\mathcal{V}, \mathcal{E})$ be a directed or undirected graph and $\mathcal{P} \subseteq \mathcal{E}$ be a subset of edges. A *vertex cover* of $\mathcal{P}$ is a subset $\mathcal{S}$ of $[1, N]$ such that for each edge $(i, j) \in \mathcal{P}$ either the source i or the sink j lies in $\mathcal{S}$. We define a *non-trivial* vertex cover to be a vertex cover $\mathcal{S}$ such that $\mathcal{S} \subseteq \mathcal{V}(\mathcal{P})$. We denote the size of a minimal vertex cover of $\mathcal{P}$ by

$$\mathsf{VC}(\mathcal{P}) := \min\{|\mathcal{S}| : \mathcal{S} \subseteq [1, N] \text{ covers } \mathcal{P}\}.$$

7.3.1 Graph Pebbling

A *pebbling configuration* on the graph $\mathcal{G}$ is a set $\mathcal{P} \subseteq \mathcal{E}$ defining the subset of pebbled edges. Let $|\mathcal{P}|$ denote the number of pebbles in the configuration and $\mathcal{V}(\mathcal{P})$ the set of nodes

[9] Since most of our ideal functionalities are implemented using random oracles, it is possible using standard tricks [IR89] to switch the order of the quantifiers and establish the stronger statement that there exists a *single* oracle P and adversary A_Q which work *for all* reductions.

involved in the pebbling. We define the *complexity* of a pebbling configuration $\mathcal{P}$ as the size of a minimal vertex cover of $\mathcal{P}$.[10] For a pebbling sequence $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\ell)$, we define $\mathsf{VC}(\boldsymbol{\mathcal{P}}) := \max_{i\in[0,L]} \mathsf{VC}(\mathcal{P}_i)$.

Let $\mathcal{P}_{\text{start}}$ denote the unique configuration with $|\mathcal{P}_{\text{start}}| = \mathsf{VC}(\mathcal{P}_{\text{start}}) = 0$, i.e., $\mathcal{P}_{\text{start}} = \emptyset$, and $\mathcal{P}_{\text{target}} = \text{in}(T) = \{(i, T) \in \mathcal{E}\}$ denote the configuration where only all the edges incident on some sink $T \in \mathcal{T}$ are pebbled. We will consider sequences of pebbling configurations $\boldsymbol{\mathcal{P}} = (\mathcal{P}_{\text{start}}, \ldots, \mathcal{P}_{\text{target}})$ where subsequent configurations have to follow the reversible edge-pebbling rules from Definition 8 in Chapter 3:

Definition 53 (Edge-pebbling)**.** An edge pebbling of a DAG $G = (\mathcal{V}, \mathcal{E})$ with *unique sink* T is a pebbling sequence $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\ell)$ with $\mathcal{P}_0 = \mathcal{P}_{\text{start}}$ and $\mathcal{P}_\tau = \mathcal{P}_{\text{target}}$, such that for all $i \in [1, \tau]$ there is a unique $(u, v) \in \mathcal{E}$ such that:

- $\mathcal{P}_i = \mathcal{P}_{i-1} \cup \{(u, v)\}$ or $\mathcal{P}_i = \mathcal{P}_{i-1} \setminus \{(u, v)\}$,
- $\text{in}(u) \subseteq \mathcal{P}_{i-1}$.

For some applications, we will actually consider the classical *reversible node-pebbling* as in [Ben89], where pebbling configurations $\mathcal{P}_i$ are subsets of *nodes*: A node is deemed pebbled whenever all ingoing edges are pebbled, and two subsequent pebbling configurations differ by a node. Since any node-pebbling sequence induces an edge-pebbling sequence, we view node-pebbling as a more restricted version of edge pebbling. The following definition of node-pebbling is equivalent to the traditional notion from [Ben89].

Definition 54 (Node-pebbling)**.** A node pebbling of a DAG $G = (\mathcal{V}, \mathcal{E})$ with *unique sink* T is a pebbling sequence $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ with $\mathcal{P}_0 = \mathcal{P}_{\text{start}}$ and $\mathcal{P}_\tau = \mathcal{P}_{\text{target}}$, such that for all $i \in [1, \tau]$ there is a unique $v \in [1, N]$ such that:

- $\mathcal{P}_i = \mathcal{P}_{i-1} \cup \text{in}(v)$ or $\mathcal{P}_i = \mathcal{P}_{i-1} \setminus \text{in}(v)$,
- for all $u \in \text{parents}(v)$: $\text{in}(u) \subseteq \mathcal{P}_{i-1}$.

Definition 55 (Configuration graph)**.** Let $G = (\mathcal{V}, \mathcal{E})$ be some graph. We define the associated *configuration graph* $\mathcal{P}^G$ as the graph that has as its vertex set all $2^{|\mathcal{E}|}$ possible pebbling configurations of G. The edge set will contain an edge between two vertices, if the transisition between the two vertices is an allowed pebbling move according to the pebbling game rules.

Note that the configuration graph depends on the pebbling game. If we consider reversible pebbling as in Definitions 53 and 54, the configuration graph is undirected.

7.4 Builder-Pebbler Game

In this work, we consider security games for multi-user schemes where an adversary can adaptively do the following actions:

[10] Recall, in Chapter 3 we used the *lateral space complexity* (Definition 10) to prove upper bounds on the loss in security for GSD. It is easy to see that our notion of pebbling complexity in terms of minimal vertex cover lower bounds the lateral space complexity.

- query for information between pairs of users,
- corrupt users and gain secret information associated to these users,
- issue a distinguishing challenge query associated to a target user of its choice,
- guess a bit $b \in \{0, 1\}$.

We consider such games as games on graphs, where users represent the nodes of the graph and edges are defined by the adversary's pairwise queries. If the pairwise information depends asymmetrically on the two users, then this is represented by the direction of the corresponding edge and after the game one can extract a directed graph structure from the transcript of the game. Here, we only consider the case of directed *acyclic* graphs, i.e., where the adversary is forbidden to query cycles. Furthermore, to avoid trivial winning strategies, the adversary must not query a challenge on a node which is reachable from a corrupt node.

To prove a scheme secure under such an adaptive game based on standard assumptions (e.g., the security of some involved primitive), a common approach is to construct a reduction that has black-box access to an adversary against the scheme and tries to use the advantage of this adversary to break the basic assumption. To this aim, the reduction has to simulate the game to the adversary and at the same time embed some challenge c on the basic assumption into its answers so that the adversary's output varies depending on this embedded challenge. Hence, the reduction might not answer all queries correctly but rather "fakes" some of the edges; such wrong answers will be represented as *pebbled* edges in the graph. However, if the reduction answers all queries connected to the challenge node independent of the challenge user's secrets, then the edge queries do not help the adversary to distinguish its challenge and its advantage in this game can be at most the advantage it has in an alternative security game where no edge queries are possible. Indistinguishability in such a weaker scenario usually follows trivially by some basic assumption.

Thus, we are interested in games that can be abstracted by the following two-player game.

Definition 56 (N- and $(N, \mathcal{G})$-Builder-Pebbler Game)**.** For a parameter $N \in \mathbb{N}$, the N-Builder-Pebbler Game is played between two players, called Builder and Pebbler, in at most $N \cdot (N-1)/2$ rounds. The game starts with an empty DAG $G = ([1, N], \mathcal{E} = \emptyset)$ and an empty set $\mathcal{P} = \emptyset$. In each round:

1. the Builder first picks an edge $e \in [1, N]^2 \setminus \mathcal{E}$ and adds it to G (i.e., $\mathcal{E} := \mathcal{E} \cup \{e\}$); the Builder is restricted to only query edges that do not form cycles; and
2. the Pebbler then either places a pebble on e (i.e., $\mathcal{P} := \mathcal{P} \cup \{e\}$) or not (i.e., $\mathcal{P}$ remains the same).

The Builder can stop the game at any point by choosing a sink in G as the challenge. This results in a *challenge* DAG $G^* = (\mathcal{V}^*, \mathcal{E}^*)$, the subgraph of G that is induced by all nodes from which the challenge is reachable.
In an $(N, \mathcal{G})$-Builder-Pebbler Game, the Builder is restricted to building graphs (isomorphic to subgraphs of) $G \in \mathcal{G}$ for a class of graphs $\mathcal{G}$.

Definition 57 (Winning condition and advantage for $(N, \mathcal{G})$-Builder-Pebbler Game)**.** Consider an $(N, \mathcal{G})$-Builder-Pebbler Game and let $G = (\mathcal{V}, \mathcal{E})$, $G^* = (\mathcal{V}^*, \mathcal{E}^*)$ and $\mathcal{P}$ be as in

Definition 56. We model the winning condition for the game through a function X that maps a graph to a collection of subsets of its own edges. We say that the Pebbler wins the $(N, \mathcal{G})$-Builder-Pebbler Game under winning condition X if the following two conditions are satisfied:

1. only edges rooted in $\mathcal{V}^*$ are pebbled, i.e. $\mathcal{P} \subseteq \{(u, v) \in \mathcal{E} \mid u \in \mathcal{V}^*\}$
2. the pebbling induced on G^* satisfies the winning condition, i.e., $\mathcal{P}|_{G^*} \in X(G^*)$.

Otherwise, the Builder is declared the winner. In case the strategies are randomised, we call the probability (over the randomness of the strategies) with which the Builder (resp., Pebbler) wins the game the *Builder's (resp., Pebbler's) advantage*, and denote it by $\beta = \beta(N)$ (resp., $\pi = \pi(N)$). Since there are no draws, we have $\beta + \pi = 1$.

Remark 10. The corresponding definitions for the N-Builder-Pebbler Game can be obtained by simply ignoring the restriction to $\mathcal{G}$.

In our setting we will be interested in functions X that output sets of vertices that represent the frontier of a cut in the configuration graph of the input.

Definition 58 (Cut function)**.** For a family $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of graphs, a function $X : \mathcal{G} \mapsto 2^{\mathcal{E}}$ is called a *cut function* if $X(G)$ is the frontier of an s-t-cut of the configuration graph $\mathcal{P}^G$ that separates $\mathcal{P}_{\text{start}}$ from $\mathcal{P}_{\text{target}}$ for any input $G \in \mathcal{G}$. For a cut function X defined on $\mathcal{G}$ and $G \notin \mathcal{G}$, we set $X(G) = \emptyset$.

7.4.1 Player Strategies

Builder strategies. As motivated in Remark 9, we will mostly be dealing with a class of Builders who play *independently* of Pebbler's strategy.[11]

Definition 59 (Oblivious Builders)**.** We say that a Builder's strategy in the $(N, \mathcal{G})$-Builder-Pebbler Game is *oblivious* if its choice of graph $G \in \mathcal{G}$ and order of edge queries are *independent* of (i.e., oblivious to) the Pebbler's strategy.

This restriction on the Builder serves two main purposes.

1. Firstly, it ensures that the Builder-Pebbler Game is not trivial for the cut functions we are interested in: otherwise, it is easy to come up with Builder strategies in which any Pebbler has advantage 0.
2. Moreover, non-oblivious Builder strategies are less interesting in our setting since they could potentially allow reductions to exploit the query behaviour of the adversary built on top of a non-oblivious Builder to gain advantage in the security game.

[11] The exact definition of the strategy will depend on the graph and the application: e.g., see Theorem 25 for an oblivious Builder strategy for paths that is later used in Corollary 6.

Pebbler strategies. Ideally, we would like to establish lower bounds that hold against all Pebblers. Since this is not always possible, we consider Builder-Pebbler Games where the Pebbler strategy is restricted. The first such class of restricted Pebbler strategies are *oblivious Pebblers*.

Definition 60 (Oblivious Pebbler)**.** We say that a Pebbler's strategy is *oblivious* if it fixes a subset of vertices $\mathcal{S} \subseteq [1, N]$ at the beginning of the game, and at the end of the game $\mathcal{S}$ is always a non-trivial vertex cover of the pebbling $\mathcal{P}$.

Note that the notion of obliviousness differs from that in Definition 59.[12] Definition 60 is motivated by oblivious reductions used in this work (see Section 7.1.1) and the goal is to capture *prior knowledge* that a Pebbler may have about the graph structure that a Builder builds during the query phase. This is captured in Definition 60 by requiring the Pebbler to commit to a non-trivial vertex cover of the pebbling configuration. This allows compressing of pebbling configurations based on the graph structure: e.g., if the Pebbler knows that the graph contains nodes with high degree and it aims to pebble all (or some) of the incident edges of such a node, it may guess this node ahead of time and then adjust its query responses assuming the guess is correct. In the known upper bounds for the applications we consider, this is used to compress the amount of information that needs to be guessed ahead of time. The fact that the vertex cover is required to be non-trivial ensures that this restriction is also non-trivial: otherwise, the Pebbler may simply output the entire set $[1, N]$. On the other hand, using a minimal vertex cover seems too strong, since we do not actually require it to prove our bounds.

The second class of restricted Pebbler strategies that we consider are *node Pebblers*. The definition is motivated by the reductions from Chapters 3 and 4 that rely on the node-pebbling game from Definition 54. Thus, the restriction placed on Pebbler is natural: whenever it places a pebble on a vertex v, it must place pebbles on all edges incident on v.

Definition 61 (Node Pebbler)**.** A Pebbler is a *node Pebbler*, if for all nodes $v \in [1, N]$ it either places a pebble on all edges incident on v or on none.

Remark 11. Note that restricting the Builder strategy does not weaken our results: we are constructing lower bounds for reductions and an oblivious Builder gives rise to oblivious adversaries. In contrast restricting to oblivious or node Pebblers does weaken the result. However, looking ahead, these restrictions allow us to prove much stronger bounds compared to an unrestricted Pebbler.

7.5 Combinatorial Upper Bounds

In this section we show upper bounds for Pebblers in the Builder-Pebbler Game by constructing Builders (potentially in a restricted Builder-Pebbler Game) and then showing that no Pebbler can have a good advantage against such a Builder. We start by considering oblivious Pebblers (Section 7.5.1), before focussing on Pebblers that may only pebble all or none of the edges incident on any node, i.e. *node Pebblers* (Section 7.5.2). Finally, we show a bound that holds for arbitrary Pebblers (Section 7.5.3).

[12] We considered changing the name of at least one of the players to make the distinction clearer, but did not find another suitable term, since both concepts capture a kind of obliviousness. So we stick with this nomenclature and simply hope it does not cause too much confusion.

7.5.1 Oblivious Pebbler

We first construct Builders that are restricted to certain families of graphs, which are common in applications. At the end of the section we consider Builders in the unrestricted Builder-Pebbler Game, which allows to deduce a much stronger bound, but which may not be as widely applicable.

Paths

For the Builder-Pebbler Game restricted to paths of length N (i.e., the class $\mathcal{C}_N$), we show that any oblivious Pebbler playing the Builder-Pebbler Game against a random Builder (which is oblivious) with a definition of cut that is closely related to pebbling lower bounds for paths (see Section 7.5.1) has advantage at most quasi-polynomial in N (Section 7.5.1). We exploit the observation that whenever the Pebbler behaves obliviously and the Builder queries edges uniformly at random, the nodes from the vertex cover will be uniformly distributed on the path. Our result matches the best known Pebbler strategy (of simply guessing the nodes in the cut) that has an advantage $\pi \geq 1/N^{\log(N)}$ up to constant factors in the exponent.

Pebbling Characteristics of Paths. To define a suitable cut in the configuration graph, we use a known lower bound on the number of pebbles needed to reversibly pebble a path [CDG01]: For any $k \geq 1$ and every pebbling sequence $\boldsymbol{\mathcal{P}}_k = (\mathcal{P}_{\text{start}}, \ldots, \mathcal{P}_k)$, where $(2^k, 2^k + 1) \in \mathcal{P}_k$, it must hold $|\boldsymbol{\mathcal{P}}_k| := \max\{|\mathcal{P}| \mid \mathcal{P} \in \boldsymbol{\mathcal{P}}_k\} \geq k + 1$. One can prove this by induction: First, note that pebbling the second edge $(2, 3)$ requires 2 pebbles. Now assume the claim is true for $k - 1$ with $k > 1$. Clearly, any valid pebbling sequence $\boldsymbol{\mathcal{P}}_k$ must contain a configuration where the 2^{k-1}th edge is pebbled for the first time, i.e., the 2^{k-1}th edge is pebbled and all subsequent edges are unpebbled. Assume $|\boldsymbol{\mathcal{P}}_k| \leq k$ and consider the following two cases: Either the 2^{k-1}th edge remains pebbled until the 2^kth edge is pebbled, which would immediately imply a pebbling strategy to pebble the 2^{k-1}th edge using only $k - 1$ pebbles – a contradiction. Or the pebble on the 2^{k-1}th edge is removed while there is at least one pebble on some subsequent edge (to guarantee progress), which would imply that the pebble on the 2^{k-1}th edge can be removed using only $k - 2$ additional pebbles – again a contradiction due to the reversible pebbling rules. This proves the claim. The above lower bound is indeed tight and a matching reversible pebbling strategy can be found, for example, in [Ben89].

In particular, for all valid edge pebbling sequences $\boldsymbol{\mathcal{P}} = (\mathcal{P}_{\text{start}}, \ldots, \mathcal{P}_{\text{target}})$ of a path on $N = 2^n + 1$ nodes, with $\mathcal{P}_{\text{start}} = \emptyset$ and $\mathcal{P}_{\text{target}}$ being the configuration where only the last edge is pebbled, there must exist a pebbling configuration $\mathcal{P} \in \boldsymbol{\mathcal{P}}$ such that $|\mathcal{P}| = \lfloor \log(N) \rfloor + 1$. Thus, we define a cut in the configuration graph as follows:

Definition 62 (Good pebbling configurations, cuts and cut function for paths)**.** We call a pebbling configuration $\mathcal{P}$ for a path $C = C_N$ of on N nodes *good* if it contains $\lfloor \log(N) \rfloor$ pebbles and there exists a valid pebbling sequence $\boldsymbol{\mathcal{P}} = (\mathcal{P}_{\text{start}}, \ldots, \mathcal{P})$ such that $|\boldsymbol{\mathcal{P}}| = \lfloor \log(N) \rfloor$. We define a *cut set* X in the configuration graph $\mathcal{P}^C$ as the set of all edges consisting of a good pebbling configuration and a configuration which can be obtained from this good configuration by *adding* one pebble (following the pebbling rules). The cut function $X_{\mathcal{C}}$ is defined as in Definition 58 as the frontier of this cut.

Remark 12. A complete characterisation of such reachable configurations is given in [LTV98]. Let the pebbles in a configuration $\mathcal{P}$ be $\{(v_i, v_i + 1)\}_{i \in [1, \log(N)]}$ for $v_i \in [0, N - 1]$. Then $\mathcal{P}$ is reachable if and only if for every $i \in [1, \log(N)]$, $\mathcal{P}$ has a pebble in the range $\{(v_i - 2^i, v_i - 2^i + 1), \ldots, (v_i - 1, v_i)\}$.

The Upper Bound. Since we consider oblivious Pebbler strategies, this means that a successful Pebbler must choose a vertex cover $\mathcal{S} \subseteq [1, N]$ such that each node in $\mathcal{S}$ is either source or sink of a pebbled edge in $\mathcal{P}$. If the adversary queries a uniformly random path on $[1, N]$, then $\mathcal{S}$ will be a uniformly random subset of nodes. Obviously, we must have $(\log(N))/2 \leq |S| \leq 2\log(N)$. In the following Lemma we bound the probability that a uniformly random subset $\mathcal{S}$ of nodes of some fixed size $s \in [(\log(N))/2, 2\log(N)]$ is a vertex cover of a good configuration $\mathcal{P}$ and $\mathcal{S}$ is a subset of the nodes $V(\mathcal{P})$ involved in $\mathcal{P}$.

Lemma 31. *Let $\mathcal{S} \subseteq [1, N]$ be a uniformly random subset of size $s \in [(\log(N))/2, 2\log(N)]$, $\sigma = \min\{s, \log(N)\}$, and P be the set of good pebbling configurations on paths on N nodes. Then*

$$\mathsf{Pr}[\exists \mathcal{P} \in P : \ \mathcal{S} \ covers \ \mathcal{P} \wedge \mathcal{S} \subseteq V(\mathcal{P})] \leq \frac{s^{2s}}{N^{s-\sigma} 2^{\sigma(\sigma+1)/2}} \leq \frac{N^{\log(\log(N))}}{N^{\log(N)/8}}.$$

Proof. We call $\mathcal{S}$ *good* if it covers a good pebbling configuration $\mathcal{P}$ and $\mathcal{S} \subseteq V(\mathcal{P})$. First, we count the number of subsets of size s which are good. To this aim, note that since we consider reversible pebbling, a configuration $\mathcal{P}$ with $|\mathcal{P}| = \log(N)$ is good if and only if all pebbles can be removed without the need of any additional pebbles.
Now, assume $\mathcal{S}$ covers a good pebbling configuration $\mathcal{P}$ and $\mathcal{S} \subseteq V(\mathcal{P})$. If $s \geq \log(N)$, then it must be the case that there are $\bar{s} \geq s - \log(N)$ pairs of nodes in $\mathcal{S}$ such that both nodes cover the same edge in $\mathcal{P}$, respectively, and one node from each pair can be removed from $\mathcal{S}$ such that the remaining set $\mathcal{S}' \subseteq S$ still covers $\mathcal{P}$. Let $\bar{s}$ be maximal with this property, hence $\mathcal{S}'$ a minimal vertex cover of $\mathcal{P}$; we denote its size by $s' = s - \bar{s}$. Clearly $s' \leq \log(N)$, and there must exist $\log(N) - s'$ nodes in $\mathcal{S}'$ which each cover two edges and the pairs of consecutive edges are pairwise disjoint.
Considering the edges in $\mathcal{P}$ to be pebbled, one pebble of each such pair of consecutive pebbles can be removed trivially from the graph. These $\log(N) - s'$ pebbles can now be used to remove further pebbles. Note, in general, using k pebbles, one can remove a pebble at distance at most 2^k from its predecessor. This in particular implies that the set $\mathcal{S}'$ must contain a pair of nodes u_1, v_1 that have distance at most $2^{\log(N)-s'}$ in the path. After removing the pebble incident on node v_1, we have one more pebble at our disposal to remove a further pebble incident on a node in $\mathcal{S}' \setminus \{v_1\}$ at distance $\leq 2^{\log(N)-s'+1}$ of its predecessor in $\mathcal{S}' \setminus \{v_1\}$ on the path. Pursuing this idea, in the kth step, there must be a node at distance $\leq 2^{k+\log(N)-s'}$ from its preceding node on the path. In total, there are $\log(N) - s'$ gaps between nodes in $\mathcal{S}'$, where one gap is of size $\in [1, 2^{\log(N)-s'}]$, another one is of size $\in [1, 2^{\log(N)-s'+1}]$, another one of size $\in [1, 2^{\log(N)-s'+2}]$, and so on, up to size $\in [1, N/2]$.[13]
In total, for the s gaps on the path between the nodes in $\mathcal{S}$ it holds: $s - \sigma \leq \bar{s}$ gaps must be of size 1, the remaining $\sigma - s'$ gaps of size 1 are in particular one gap of size $\in [1, 2^{\log(N)-\sigma}]$, one of size $\in [1, 2^{\log(N)-\sigma+1}]$, and so on, up to size $\in [1, 2^{\log(N)-s'-1}]$. For the remaining s' gaps, as stated above, there must be one gap of size $\in [1, 2^{\log(N)-s'}]$, one of size $\in [1, 2^{\log(N)-s'+1}]$, up to size $\in [1, N/2]$. Thus, independent of $\bar{s}$, there must be $s - \sigma$ gaps of size 1 and σ gaps of sizes $\in [1, 2^{\log(N)-\sigma+k-1}]$ for $k \in [1, \sigma]$, respectively.
Thus, we can upper bound the number of good subsets of size s as the number of possible subsets of s nodes having the required gap sizes on the path as discussed. Of course, the s gaps do not need to be in order, so we get an upper bound on the number of different good

[13] Note, we also consider the distance between the source node and the first node in $\mathcal{S}$ on the path as a gap.

subsets $\mathcal{S}$ by

$$\#\text{ good subsets } \leq s! \cdot \prod_{k=0}^{\sigma-1} 2^{\log(N)-\sigma+k} \leq s^s \cdot 2^{\sum_{k=\log(N)-\sigma}^{\log(N)-1} k} = s^s \cdot 2^{\sigma(2\log(N)-\sigma-1)/2}.$$

On the other hand, the total number of subsets of s nodes is $\binom{N}{s}$. Thus, we can upper bound the probability of $\mathcal{S}$ being good by

$$\mathsf{Pr}[\mathcal{S} \text{ is good}] \leq \frac{s^s \cdot 2^{\sigma(2\log(N)-\sigma-1)/2}}{\binom{N}{s}} \leq \frac{s^s \cdot 2^{\sigma\log(N)-\sigma(\sigma+1)/2} \cdot s^s}{N^s} \leq \frac{s^{2s}}{N^{s-\sigma}2^{\sigma(\sigma+1)/2}}.$$

This upper bound is maximal when $s = \log(N)/2$, where it attains

$$\frac{(\log(N)/2)^{\log(N)}}{2^{(\log(N)/2)\cdot(\log(N)/2+1)/2}} \leq \frac{N^{\log(\log(N))}}{N^{\log(N)/8}}.$$

On the other hand, the probability of $\mathcal{S}$ being good is 0 whenever it has size $< \log(N)/2$ or $> 2\log(N)$. The claim follows. □

Lemma 31 immediately allows us to prove the following upper bound on the advantage $\pi(N)$ of an oblivious Pebbler whenever we restrict the Builder-Pebbler Game to paths.

Theorem 25 (Combinatorial Upper Bound for Paths)**.** *The advantage of any oblivious Pebbler against a random Builder in the $(N, \mathcal{C}_N)$-Builder-Pebbler Game with the winning condition $X_{\mathcal{C}}$ defined as in Definition 62 is at most*

$$\pi \leq 1/N^{\log(N)/8-\log(\log(N))}.$$

Proof. Consider the following random Builder strategy B.

1. Pick a path P_N uniformly at random from $\mathcal{P}_N$ the set of all paths of length N.
2. Query the edges of P_N, one at a time in random order.
3. Challenge the (only) sink of P_N.

Since the Pebbler is oblivious it has to commit to some vertex cover $\mathcal{S} \subseteq [1, N]$ in the beginning of the game. Since B queries edges uniformly at random, $\mathcal{S}$ is a uniformly random subset of $[1, N]$. Thus, by Lemma 31, the probability that $\tilde{\mathcal{P}}$ is good is at most

$$\frac{N^{\log(\log(N))}}{N^{\log(N)/8}} = \frac{1}{N^{\log(N)/8-\log(\log(N))}}.$$

This proves the theorem. □

Binary In-Trees

In the case we restrict the Builder-Pebbler Game to binary trees directed from the leaves to the root – short, "in-trees"; but we will simply refer to them as binary trees in this entire section – we show that any oblivious Pebbler playing the Builder-Pebbler Game against a random Builder with a definition of cut that is again related to pebbling lower bounds for trees (see Section 7.5.1) has advantage at most quasi-polynomial in N (Section 7.5.1). As a warm up, we analyze in Section 7.5.1 the advantage of an oblivious Pebbler when the size of the vertex cover is bounded (i.e, $o(N)$ to be precise). We then extend this to arbitrary oblivious strategies for Pebbler (Section 7.5.1). The main idea is to borrow ideas from pebbling lower bounds for binary trees as described in the introduction (and recalled below).

Pebbling Characteristics of Binary Trees. It is known that the number of pebbles that are needed to pebble a perfect binary tree B_n of depth n, and therefore of size $N = 2^{n+1} - 1$, is at least n, and the argument is as follows (refer to [Sav98] for example). Consider a pebbling sequence for a perfect binary tree: at the beginning of the sequence *none* of the 2^n paths from the root to the leaves carries a pebble, whereas at the end of the sequence (at which point the edges incident on the root carry a pebble) *all* the paths from the leaves to the root carry a pebble. Furthermore, only new pebbles outgoing from leaves decrease the number of paths that carry a pebble, because a pebble can only be placed on an inner edge if both edges incident on its source are already pebbled. Hence, all paths through the source of this this inner edge already carry a pebble. So any pebbling move can only decrease the number of paths that carry a pebble by 1. Therefore there has to exist two consecutive configurations in the pebbling sequence such that in the first configuration there exists a path that does not carry a pebble but in the next configuration every path carries a pebble. At this point at least all the edges incident on this path which do *not* lie on the path themselves need to be pebbled, and in particular there exists a pebbling configuration where there are at least $\log(N)$ pebbles. Such pairs of configurations serve as the cut for the winning condition. A formal definition follows.

Definition 63 (Good pebbling configurations, cuts and cut function for binary trees)**.** Let $\boldsymbol{\mathcal{P}}$ be the set of pebbling configurations on B_n such that B_n contains at least one path from a leaf to the root that does not carry a pebble, i.e. all edges on this path are unpebbled. As the cut-set on (the configuration graph of) B_n we choose $X = \{(\mathcal{P}_i, \mathcal{P}_j) \mid \mathcal{P}_i \in \boldsymbol{\mathcal{P}} \wedge \mathcal{P}_j \notin \boldsymbol{\mathcal{P}}\}$. Note that any $\mathcal{P}_i$ with $(\mathcal{P}_i, \mathcal{P}_j) \in X$ for some $\mathcal{P}_i$ must have exactly one path from some leaf to the root not carry a pebble while every other path must carry a pebble. The cut function $X_{\mathcal{B}}$ is defined as in Definition 58 as the frontier of this cut.

Warm-up: Upper Bound for Bounded Vertex Cover. Consider an oblivious Pebbler that selects at most s (random) vertices on the binary tree as the vertex cover. (Note that since the Pebbler is oblivious and the Builder picks a uniformly random permutation of the graph, we can view any oblivious Pebbler as selecting the vertices in the cover at random.) For the ease of analysis, we will consider a slightly different Pebbler which – instead of selecting s vertices at random – will include each vertex in the cover with probability $\alpha := s/N$. We first show in Lemma 32 that this cannot decrease the success probability too much, so any super-polynomial lower bound we obtain in this way holds in general.

Lemma 32. *Let* P_s *be a Pebbler that selects* s *vertices at random and let* $\mathsf{P}_{\alpha(s)}$ *be a Pebbler that behaves exactly like* P_s *but for every one of the* N *vertices chooses to include it in the vertex cover i.i.d. with probability* $\alpha(s) = s/N$*. Then for any event* E *over the output of* P_s *we have* $\mathsf{Pr}[\mathsf{P}_s \to E] = O(\sqrt{N})\mathsf{Pr}[\mathsf{P}_{\alpha(s)} \to E]$*.*

Proof. Let L be the event that $\mathsf{P}_{\alpha(s)}$ selects exactly s vertices. Then we have $\mathsf{Pr}[\mathsf{P}_s \to E] = \mathsf{Pr}[\mathsf{P}_{\alpha(s)} \to E \mid L]$. On the other hand, we have

$$\mathsf{Pr}[\mathsf{P}_{\alpha(s)} \to E] = \mathsf{Pr}[\mathsf{P}_{\alpha(s)} \to E \mid L]\mathsf{Pr}[L] + \mathsf{Pr}[\mathsf{P}_{\alpha(s)} \to E \mid \bar{L}]\mathsf{Pr}[\bar{L}]$$

where $\bar{L}$ is the complementary event to L. This implies

$$\mathsf{Pr}[\mathsf{P}_s \to E] \leq \mathsf{Pr}[\mathsf{P}_{\alpha(s)} \to E]/\mathsf{Pr}[L].$$

It remains to bound $\mathsf{Pr}[L]$ from below:

$$\mathsf{Pr}[L] = \binom{N}{s}\left(\frac{s}{N}\right)^s\left(\frac{N-s}{N}\right)^{N-s} = \frac{N!}{N^N}\frac{s^s}{s!}\frac{(N-s)^{N-s}}{(N-s)!} \geq \sqrt{\frac{N}{2\pi e s(N-s)}}.$$

□

Theorem 26 (Combinatorial Upper Bound for Binary Trees: Bounded VC)**.** *Let $\mathcal{B}_n$ be the class of perfect binary trees of depth n and size $N = 2^{n+1}-1$. Then any oblivious Pebbler* P *in the $(N, \mathcal{B}_n)$-Builder-Pebbler Game with perfect binary trees with cut $X_{\mathcal{B}}$ defined as in Definition 63 has an advantage of at most*

$$\pi \leq \begin{cases} 1/N^{\log(N)} & \text{for } s = o(N) \\ 1/N^{\omega(1)} & \text{for } s = N^{\epsilon} \text{ with } \epsilon < 1 \text{ constant.} \end{cases}$$

against a random Builder B*.*

Proof. Note first that since B queries a random graph $B_n \in \mathcal{B}_n$, one can view P_s as choosing the s vertices in the cover uniformly at random. By Lemma 32 we can instead bound the advantage of $\mathsf{P}_{\alpha(s)}$, which chooses for each vertex i.i.d. if it will be included in the cover with probability $\alpha = s/N$.

Fix a path p from a leaf to the root in B_n. Define $\boldsymbol{\mathcal{P}}_p \subset \boldsymbol{\mathcal{P}}$ to be the set of configurations in which p does not carry a pebble but every other path does. In the following we say that a subtree is *covered* by $\mathcal{S}$, if there exists a configuration $\mathcal{P}$ in which all paths from the leaves to the root of this subtree carry a pebble, such that $\mathcal{S}$ is a vertex cover of $\mathcal{P}$ and $\mathcal{S} \subset \mathcal{P}$. Let $P(d)$ denote the probability that a perfect binary tree of depth d is covered when vertices are included in the cover independently using coin toss of bias $\alpha = s/N$. We argue via induction that $P(d) \leq 2\alpha$. For the base case, note that $P(1) = \alpha + (1-\alpha)\alpha^2 \leq 2\alpha$. Suppose that the hypothesis is true for binary tree of depth $d-1$. It is not hard to see that

$$P(d) = \alpha + (1-\alpha)P(d-1)^2.$$

It follows that $P(d) \leq \alpha + (1-\alpha)4\alpha^2$, and it suffices to show that

$$(1-\alpha)4\alpha^2 \leq \alpha \Leftrightarrow (1-\alpha)\alpha \leq 1/4.$$

This is indeed true since $(1-\alpha)\alpha$ is a quadratic polynomial which is maximized at $\alpha = 1/2$.

In order for a configuration to be in $\boldsymbol{\mathcal{P}}_p$, all subtrees that are rooted in the copath of p must be covered by the selected vertex cover or the parent in the path must be in the vertex cover. The probability of this is $\leq \alpha + 2\alpha = 3\alpha$. Finally, since the vertices involved in each subtree are disjoint, we get that the probability of a vertex cover that is minimal for some configuration in $\boldsymbol{\mathcal{P}}_p$ is less than

$$\prod_{i=1}^{n-1} 3\alpha = (3\alpha)^{n-1}$$

where n, if you recall, is the depth of B_n.

By applying the union bound, we have that the probability that there exists *some* unpebbled path is at most $N/2 \cdot (3\alpha)^{n+1}$. It follows that $\Lambda \geq 2/(N \cdot (3\alpha)^{n+1})$, which is quasi-polynomial when $s = N^{\epsilon}$ for a constant $\epsilon < 1$, and super-polynomial when $s = o(N)$. □

Upper Bound for Unbounded Vertex Cover. Unfortunately, the above Builder strategy does not work when the Pebbler is allowed an unbounded number of vertices in the cover: in particular, in case the bias $\alpha = 1/2$ — in which case it places around $N/2$ pebbles — it gets into the cut with high probability. Thus, we need to somehow limit the number of pebbles that the Pebbler places, and this is accomplished by adding a second binary tree in the game. In the new strategy, the Builder randomly queries *two* binary trees and then proceeds to challenge one of these trees picked uniformly at random; Recall that if any edge in the other binary tree is pebbled, the Pebbler immediately loses. In case the Pebbler places too many pebbles, it is likely that it gets caught in this process. We show in the analysis that this intuition is in fact correct and consequently we obtain a tighter upper bound.

Theorem 27 (Combinatorial upper bound for binary trees: Unbounded VC)**.** *Let $\mathcal{B}_n$ be the class of perfect binary trees of depth n and size $N = 2^{n+1} - 1$. Then any oblivious Pebbler* P_s *which commits to a vertex cover of bounded size s in the $(N, \mathcal{B}_n)-$Builder-Pebbler Game with the cut function $X_{\mathcal{B}}$ defined as in Definition 63 has an advantage of at most*

$$\pi \leq 1/N^{\log(N)-\log(\log(N))}$$

against a random Builder B*.*

Proof. The random Builder B plays the Builder-Pebbler Game on $\mathcal{B}_n$ as follows: it picks $B_n \in \mathcal{B}_n$ at random, queries all edges except for the two edges incident on the root uniformly at random, and then uniformly at random challenges the root of one of the two binary subtrees ($\mathcal{B}_{n-1,b}$). Again, as in Theorem 26, by Lemma 32 we can bound the advantage of a Pebbler $\mathsf{P}_{\alpha(s)}$, which chooses for each vertex i.i.d. if it will be included in the cover with probability $\alpha = s/N$. Clearly, such a Pebbler has probability $(1-\alpha)^{N/2}$ of not selecting any vertex in $B_{n-1,1-b}$ (note that this is a requirement, since by definition of oblivious Pebblers any node in the vertex cover must be adjacent to at least one pebbled edge and there must not be any pebbled edges in the non-challenge part of the graph). By combining this with the bound obtained in Theorem 26, the probability of $\mathsf{P}_{\alpha(s)}$ selecting a vertex cover that is minimal for a configuration that is in X and is entirely unpebbled in $B_{n-1,1-b}$ is less than

$$\frac{N}{2} 3^{n-1} (1-\alpha)^{N/2} \alpha^{n-1}.$$

As a function of α, this expression is maximized for $\alpha = 2n/(N+2n)$ and yields the bound

$$N^{-\log(N)+\log(\log(N/2))+o(1)}.$$

□

Unrestricted Games

In the following we prove an almost exponential upper bound on the advantage of oblivious Pebblers in the Builder-Pebbler Game on complete graphs. Obviously, this implies a subexponential upper bound for oblivious Pebblers whenever the Builder is not restricted at all and, in particular, can query a complete graph.

Pebbling Characteristics of Complete Graphs. The best known pebbling strategy $\boldsymbol{\mathcal{P}} = (\mathcal{P}_{start}, \ldots, \mathcal{P}_{\text{target}})$ for a complete graph K_N of size N has vertex cover $\mathsf{VC}(\boldsymbol{\mathcal{P}}) = N/2 + 1$, which implies an exponential upper bound. Note, this is not trivial since the complete graph

has VC-complexity $N-1$. The strategy works as follows: First, greedily pebble all edges connected to the first half $[1, N/2]$ of the nodes in topological order; this can trivially be done at VC-complexity $N/2$. Next, unpebble all edges *within* the first half starting from those incident on node $N/2$ up to those on node 2; this still has VC-complexity $N/2$ since only edges were removed. At this point, all edges from $[1, N/2]$ to $[N/2+1, N]$ are pebbled, but there are no pebbles within either part of the graph; this configuration can be covered by the set $[1, N/2]$, but also by $[N/2+1, N]$ which will be a minimal cover for all subsequent configurations. Now, pebble all edges within the second half starting with those outgoing from node $N/2+1$ up to node $N-1$, which can be done since all ingoing edges from the first half are already pebbled; all these configurations can be covered by the set $[N/2+1, N]$. Finally, unpebble all edges not incident on N by following the sequence in reverse order, keeping N in each minimal vertex cover. This gives a valid pebbling strategy with VC-complexity $N/2+1$.

Unfortunately, our lower bound doesn't match this upper bound, but clearly gives a nontrivial result as stated in the lemma below.

Lemma 33 (Lower bound on VC-complexity of complete graphs)**.** *Let $\boldsymbol{\mathcal{P}}_N = (\mathcal{P}_{\text{start}}, \ldots, \mathcal{P}_{\text{target}})$ be a valid (edge-)pebbling sequence of the complete graph K_N of size N. Then*

$$\mathsf{VC}(\boldsymbol{\mathcal{P}}_N) \geq \sqrt{N} - 1.$$

Proof. We argue via induction on N. For $N = 1$, the claim is trivially true. Now, assume it holds for all $N' < N$. Let $\boldsymbol{\mathcal{P}}$ be a minimal (w.r.t. VC-complexity) pebbling sequence. W.l.o.g., we can assume that $\boldsymbol{\mathcal{P}}$ is *reduced* and, in particular, edges incident on N are never unpebbled again. Let $\mathcal{P}^*$ be the first configuration in $\boldsymbol{\mathcal{P}}$ where an edge (i^*, N) incident on N is pebbled and S^* be a minimal vertex cover of $\mathcal{P}^*$. If $\mathsf{VC}(\mathcal{P}^*) \geq \sqrt{N} - 1$ the claim trivially follows from $\mathsf{VC}(\boldsymbol{\mathcal{P}}_N) \geq \mathsf{VC}(\mathcal{P}^*)$. Thus, in the following we consider the case $|S^*| = \mathsf{VC}(\mathcal{P}^*) < \sqrt{N} - 1$. When we remove the set S^* as well as the two nodes i^* and N (where at least one of them is contained in S^*) from the graph K_N, we end up with a complete (sub)graph K^* which is entirely unpebbled and will be pebbled during the configurations $\mathcal{P}^*, \ldots, \mathcal{P}_{\text{target}}$. It holds

$$N - 2 \;\geq\; |V(K^*)| \;\geq\; |K_N| - |S^*| - 1 > N - (\sqrt{N} - 1) - 1 = N - \sqrt{N}.$$

By induction hypothesis, any valid pebbling sequence on K^* has VC-complexity at least $\sqrt{|V(K^*)|} - 1 \geq \sqrt{N - \sqrt{N}} - 1$; this in particular also holds for the pebbling sequence on K^* induced by $\boldsymbol{\mathcal{P}}$. Since the edge (i^*, N) remains pebbled throughout $\mathcal{P}^*, \ldots, \mathcal{P}_{\text{target}}$ and is node-disjoint with K^*, it follows

$$\mathsf{VC}(\boldsymbol{\mathcal{P}}_N) \geq \mathsf{VC}(\boldsymbol{\mathcal{P}}) \geq (\sqrt{N - \sqrt{N}} - 1) + 1 = \sqrt{N - \sqrt{N}}.$$

The claim now follows since $\sqrt{N - \sqrt{N}} \geq \sqrt{N} - 1$ for all $N \geq 1$. □

This also yields the following definition of good configuration for the complete graph.

Definition 64 (Good pebbling configurations, cuts and cut function for complete graphs)**.** We call a pebbling configuration $\mathcal{P}$ for the complete graph K_N of size N *good* if the VC-complexity of $\mathcal{P}$ is $\sqrt{N} - 2$ and there exists a valid pebbling sequence $\boldsymbol{\mathcal{P}} = (\mathcal{P}_{\text{start}}, \ldots, \mathcal{P})$ such that the VC-complexity of the sequence $\mathsf{VC}(\boldsymbol{\mathcal{P}}) \leq \sqrt{N} - 2$. As the cut-set on (the configuration graph of) K_N we choose the X to be defined as the set of pairs $(\mathcal{P}_i, \mathcal{P}_j)$ such that $\mathcal{P}_i$ is good, $\mathcal{P}_j$ is not good, and $\mathcal{P}_j$ differs from $\mathcal{P}_i$ in one valid pebbling step. The cut function $X_{\mathcal{K}}$ is defined as in Definition 58 as the frontier of this cut.

The Upper Bound. Lemma 33 implies the following upper bound on the advantage π for oblivious Pebblers against a random Builder on the Builder-Pebbler Game played on a complete challenge graph.

Theorem 28 (Combinatorial upper bound for complete graphs)**.** *Let $\mathcal{K}_N$ denote the class of complete directed graphs on N vertices. Then any oblivious Pebbler in the $(N, \mathcal{K}_N)$-Builder-Pebbler Game with the cut function $X_{\mathcal{K}}$ defined in Definition 64 has advantage at most*

$$\pi \leq e^{-2(\sqrt{N/(e^3)}-1)}.$$

against a random Builder B*.*

Proof. We use the following random Builder: the graph structure B queries consists of two complete directed graphs of sizes n and $N - n$, respectively, where we will define n later in this proof. Since, by assumption, the reduction committed to a non-trivial vertex cover $S \subseteq [1, N]$ in the beginning of the game and B chose a permutation of $[1, N]$ independently and uniformly at random, the probability that S lies completely in the first part of the graph is at most

$$\frac{\binom{n}{\sqrt{n}-1}}{\binom{N}{\sqrt{n}-1}} \leq \left(\frac{n}{\sqrt{n}-1}\right)^{\sqrt{n}-1} \left(\frac{(\sqrt{n}-1)\cdot e}{N}\right)^{\sqrt{n}-1} = \left(\frac{ne}{N}\right)^{\sqrt{n}-1}.$$

By computing the derivative of the latter function one finds that it takes its minimum close to $n = N/e^3$, hence B will use this value for n. Thus, $\pi \leq e^{-2(\sqrt{N/(e^3)}-1)}$. This proves the claim. □

7.5.2 Node Pebbler

Here, we consider *node Pebblers* as defined in Definition 61, i.e., the Pebbler is only allowed to either pebble *all* ingoing edges to a node, or *none* as in the node-pebbling game (Definition 54). As we will see in Section 7.9, for certain applications it is sufficient to restrict to this class of Pebblers. Looking ahead, since node-pebbling reductions are a subclass of edge-pebbling reductions (and node pebbling strategies are a subclass of all pebbling strategies), all previous results carry over. For certain graphs of high indegree, however, we will prove much stronger bounds. These are stronger not only quantitatively, but also qualitatively as they will lead to cryptographic lower bounds which hold for arbitrary (potentially non-oblivious and rewinding) black-box reductions.

Complete Graphs

For node Pebblers in an unrestricted Builder-Pebbler Game, we prove an exponential upper bound on the Pebbler's advantage. To define a suitable cut, we exploit the crucial difference between edge and node pebbling in terms of VC-complexity regarding graphs of high indegree: Let u be an intermediate node which has high indegree in the challenge graph. In the edge pebbling game, to be able to pebble an edge (u, v), we need to have all edges incident on u pebbled; there might be up to N edges involved but, however, one can cover all these edges with the single node u. On the other hand, in the node pebbling game, to pebble node u, all the parent *nodes* need to be pebbled and, in general, the only way for the reduction to get into this configuration is to guess all parents correctly. This is formalised in the following definition and theorem.

Definition 65. For a node $v \in \mathcal{V}$, let the *reachability graph* $\mathcal{S}_v \subseteq G$ be the subgraph induced by the nodes in $\mathcal{V}$ that can be reached from v (but not v itself). Furthermore, define the *level 2 predecessor graph* $\mathcal{P}_v^2 \subseteq G$ as the subgraph induced by all the nodes in $\mathcal{V}$ from which v can be reached through a path of length at most 2 (but again not v itself). Finally, for a graph $G = (\mathcal{V}, \mathcal{E})$ define $D(G) = \max\{|E_d| \mid E_d \subseteq \mathcal{E} \wedge |\{u \mid (u,v) \vee (v,u) \in E_d\}| = 2|E_d|\}$ to be the maximum number of pairwise disjoint edges in G.

Theorem 29. *For any graph family $\mathcal{G}$ containing all graphs isomorphic to some connected DAG $G = (\mathcal{V}, \mathcal{E}) \in \mathcal{G}$, there exists a cut function X and a Builder* B *such that any (not necessarily oblivious) node Pebbler* P *has advantage at most*

$$\pi \leq \left(\max_{v \in \mathcal{V}} \binom{D(\mathcal{S}_v) + D(\mathcal{P}_v^2)}{D(\mathcal{P}_v^2)} \right)^{-1}$$

in the $(N, \mathcal{G})$-Builder-Pebbler Game.

We remark that for any $v \in \mathcal{V}$, $D(\mathcal{P}_v^2)$ must be smaller than or equal to the in-degree of v. So, Theorem 29 only yields interesting results for graphs with large degree (but not all of them). Furthermore, if $\mathcal{S}_v$ and $\mathcal{P}_v^2$ contain long paths, then they have many disjoint edges.

Proof. Let v be such that it maximizes the quantity in the theorem. We define a cut S on $\mathcal{P}^G$ as containing all configurations where v and $\mathcal{S}_v$ are entirely unpebbled. The cut function X is now defined as the frontier of that cut (after applying the isomorphism). Note that for any configuration in $X(G)$ all edges in $\mathcal{P}_v^2$ are pebbled, while all edges in $\mathcal{S}_v$ are unpebbled. The Builder B picks a random graph G' in $\mathcal{G}$ and first queries for the disjoint edges in $\mathcal{P}_v^2$ and $\mathcal{S}_v$ in a random order. Note that the Pebbler P has no information about which edge is in $\mathcal{P}_v^2$ and which is in $\mathcal{S}_v$. Accordingly, the probability of the challenge graph being in $X(G')$ at the end of the query phase is at most

$$\binom{D(\mathcal{S}_v) + D(\mathcal{P}_v^2)}{D(\mathcal{P}_v^2)}^{-1}. \tag{7.1}$$

Note that the above argument still works if we let B send all the queries of the first phase (i.e., randomly permuted $\mathcal{P}_v^2$ and $\mathcal{S}_v$ edges) at once: as they are disjoint, getting them all at once is of no help to P for guessing whether an edge belongs to $\mathcal{P}_v^2$ or $\mathcal{S}_v$. As for a single query there's no distinction between an oblivious or non-oblivious Pebbler (as there's no second query that could depend on the answer to the first), this upper bound applies to non-oblivious Pebblers. □

Corollary 5. *For any graph family $\mathcal{G}$ containing all graphs isomorphic to the complete directed graph K_N, there exists a cut function X and a Builder* B *such that any (not necessarily oblivious) node Pebbler* P *has advantage at most*

$$\pi \leq 2^{-\Omega(N)}$$

in the $(N, \mathcal{G})$-Builder-Pebbler Game.

Proof. Invoke Theorem 29 with $v = N/2$. Note that $\mathcal{P}_v^2$ is the entire subgraph induced by $[1, N/2 - 1]$, and similarily $\mathcal{S}_v$ is the entire subgraph induced by $\{N/2 + 1, \ldots, N\}$. Both $\mathcal{P}_v^2$

and $\mathcal{S}_v$ have about $N/4$ disjoint edges (simply pick every second edge along the longest path), so by Theorem 29 any node Pebbler P has advantage at most

$$\pi \leq \binom{N/2}{N/4}^{-1} \leq 2^{-\Omega(N)}.$$

□

7.5.3 Unrestricted Pebbler

Trees

In this section we prove a first combinatorial upper bound for unrestricted – i.e., non-oblivious – Pebblers in the Builder-Pebbler Game. While our upper bound on the advantage of unrestricted Pebblers is significantly weaker than the result for oblivious Pebblers, it is still non-trivial. It relies on a generalization of the known pebbling characteristics for paths from Section 7.5.1.

Generalized Pebbling Characteristics of Paths. Let $k \in [1, N]$ be arbitrary. We prove that any pebbling sequence on a path of length N must contain a pebbling configuration such that $\lfloor \log(\lceil N/k \rceil) \rfloor + 1$ of the $\lceil N/k \rceil$ subpaths of length $\leq k$ contain at least one pebble respectively. Note, for $k = 1$ this result is already known and was proven in Section 7.5.1. Assume, for contradiction, that there exists a $k > 1$ and a valid pebbling strategy $\boldsymbol{\mathcal{P}}$ for paths of length N such that the claim was false. Then this strategy implies a pebbling strategy $\boldsymbol{\mathcal{P}}'$ of complexity less than $\lfloor \log(\lceil N/k \rceil) \rfloor + 1$ for paths of length $\lceil N/k \rceil$ as follows: For each pebbling configuration $\mathcal{P}$ in $\boldsymbol{\mathcal{P}}$, define $\mathcal{P}'$ in $\boldsymbol{\mathcal{P}}'$ to contain a pebble on the ith edge if the ith subpath of $\mathcal{P}$ contains a pebble. Cancelling redundant steps in $\boldsymbol{\mathcal{P}}'$, i.e., configurations that equal the preceding configuration in the sequence, implies a valid pebbling sequence of complexity less than $\lfloor \log(\lceil N/k \rceil) \rfloor + 1$ for paths of length $\lceil N/k \rceil$ – a contradiction.

We will use the following definition of k-cuts for paths matching this generalized pebbling lower bound.

Definition 66 (k-good pebbling configurations, k-cuts and k-cut function for paths)**.** For $k \in \mathbb{N}$ we call a pebbling configuration $\mathcal{P}$ for a path $C = C_N$ on N nodes *k-good* if $\lfloor \log(\lceil N/k \rceil - 1) \rfloor$ of the $\lceil N/k \rceil - 1$ non-source subpaths of C of length $(\leq) k$ contain at least one pebble respectively[14], and there exists a valid pebbling sequence $\boldsymbol{\mathcal{P}} = (\mathcal{P}_{\text{start}}, \ldots, \mathcal{P})$ such that in all configurations in $\boldsymbol{\mathcal{P}}$ at most $\lfloor \log(\lceil N/k \rceil - 1) \rfloor$ of the subpaths simultaneously carry a pebble. We define a *k-cut set* X in the configuration graph $\mathcal{P}^C$ as the set of all edges consisting of a k-good pebbling configuration and a configuration which can be obtained from this good configuration by *adding* one pebble (following the pebbling rules) in a previously unpebbled subpath. The k-cut function $X_{\mathcal{C},k}$ is defined as in Definition 58 as the frontier of this cut.

The Upper Bound. The Builder strategy is to query a (polynomial-sized) subgraph of an exponential-sized tree of *outdegree* $\delta_{out} \geq 2$, so that in order to pebble any edge in the final challenge path the Pebbler has to guess one out of many source nodes at the same depth in the tree (see Figure 7.3).

[14] For technical reasons, we exclude the first subpath of length k in C.

Theorem 30 (Combinatorial upper bound for unrestricted Pebblers)**.** *Let $\mathcal{G}$ be the family of directed trees on $N = 2^n$ nodes (with $n \in \mathbb{N}$). Then there exists a Builder strategy querying a challenge path $G^* \in \mathcal{C}_{\sqrt{N}}$, such that the advantage of* any *Pebbler against this Builder in the $(N, \mathcal{G})$-Builder-Pebbler Game with the winning condition $X_{\mathcal{C}_{\sqrt{N}}}$ defined as in Definition 62 is at most*

$$\pi \leq 1/N^{\log(N)/8}.$$

Let $\mathcal{G}_2 \subset \mathcal{G}$ be the subset of graphs in $\mathcal{G}$ of bounded outdegree $\delta_{out} = 2$. Then there exists a Builder strategy querying a challenge path $G^* \in \mathcal{C}_{\sqrt{N}}$, such that the advantage of any *Pebbler against Builder in the $(N, \mathcal{G}_2)$-Builder-Pebbler Game with the winning condition $X_{\mathcal{C}_{\sqrt{N}},k}$ for $k = \log(N)/4$ defined as in Definition 66 is at most*

$$\pi \leq 1/N^{\log(N)/8 - \log(\log(N))/4}.$$

Proof. We define a Builder strategy B for graph family $\mathcal{G}_{\delta_{out}}$ of outdegree bounded by δ_{out} as follows: First, B chooses a source node in $[1, N]$ uniformly at random. It then proceeds in $D = N/{\delta_{out}}^{2k}$ rounds (where k is the 'overlap parameter' and will be specified later), increasing the current graph's depth by 1 in each round. In each round $R \leq 2k$ and each round $R \not\equiv 1 \mod k$, for all sinks at depth $R-1$ in the current graph B queries δ_{out} outgoing edges respectively. Note, after the first $2k$ rounds, B's queries form a δ_{out}-regular tree directed from root to leaves, with ${\delta_{out}}^{2k}$ sinks at depth $2k$ (see Figure 7.3). For all rounds such that $R > 2k$ and $R \equiv 1 \mod k$, the Builder B first chooses an integer $i \in [1, {\delta_{out}}^{k}]$ and then only queries edges outgoing from the ith batch of ${\delta_{out}}^{k}$ sinks at depth $R-1$. Finally, B chooses the target node uniformly at random from the ${\delta_{out}}^{2k}$ sinks at depth $D = N/{\delta_{out}}^{2k}$ (see Figure 7.3).

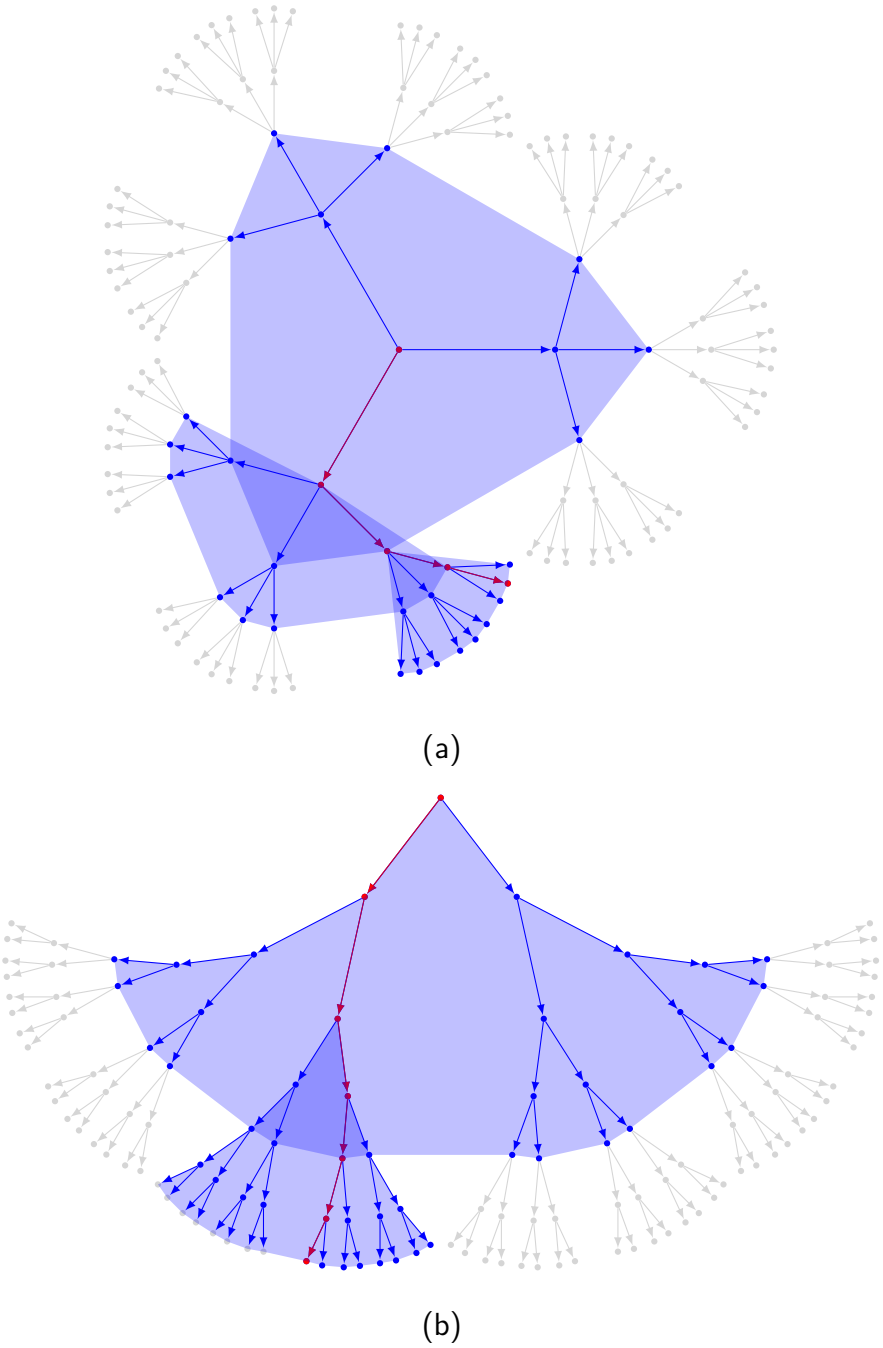
First note that B's queries involve less than $D \cdot {\delta_{out}}^{2k} = N$ nodes and the challenge graph forms a path of length D. To win the game, the Pebbler needs to place at least one pebble on $\lfloor \log(\lceil D/k \rceil - 1) \rfloor$ of the disjoint subpaths of length k in the challenge path respectively. But whenever it wants to place a pebble in a subpath starting from depth $i \cdot k$ with $i \geq 1$, the Pebbler has to at least guess which of the ${\delta_{out}}^{k}$ sources of edges at depth $i \cdot k$ will end up in the challenge graph. Since this choice is made uniformly at random by the Builder B only after all queries at depth $(i+1) \cdot k$ were made, the advantage of the Pebbler to correctly pebble an edge in the subpath sourced at depth $i \cdot k$ is at most $1/{\delta_{out}}^{k}$. Since this bound holds also conditioned on the event that previous guesses were done correctly, and to win the game, the Pebbler has to pebble $\lfloor \log(\lceil D/k \rceil - 1) \rfloor$ subpaths of the challenge path, we obtain

$$\pi \leq 1/{\delta_{out}}^{k \cdot \lfloor \log(\lceil D/k \rceil - 1) \rfloor}. \tag{7.2}$$

Now, for the graph family $\mathcal{G}$ of unbounded outdegree, we set $\delta_{out} = N^{1/4}$ and $k = 1$ to obtain $D = \sqrt{N}$ and hence $\pi \leq 1/N^{1/4 \log(\sqrt{N})} = 1/N^{\log(N)/8}$ (e.g., Figure 7.3.(a)). For $\delta_{out} = 2$, on the other hand, we set $k = \log(N)/4$ to obtain $D = \sqrt{N}$ and $\pi \leq 1/N^{1/4(\log(\sqrt{N}) - \log(\log(N)/4))} = 1/N^{\log(N)/8 - \log\log(N)/4}$ (e.g., Figure 7.3.(b)). □

7.6 Cryptographic Lower Bound I: Generalized Selective Decryption

The generalized selective decryption game (GSD) was already introduced in Section 3.3. In this section, we interpret the lower bounds from Section 7.5.1 and 7.5.3 for GSD. Then, in Section 7.6.2, we define a public-key analogue of GSD, where PKE is used instead of SKE



(a)

(b)

Figure 7.3: (a) Highlighted in blue is a regular tree of out-degree $\delta_{out} = 3$ and depth $D = 4$ that could result from the Builder strategy in Theorem 30 with overlap parameter $k = 1$. It is a subgraph of the perfect regular tree of out-degree $\delta_{out} = 3$ and depth $D = 4$ which is in the background in grey. The challenge path is highlighted in red. (b) Similar to (a), but with $\delta_{out} = 2$, depth $D = 6$ and overlap parameter $k = 2$ and the supergraph is a perfect binary tree. In both (a) and (b), the perfect subgraphs of depth $2k$ and out-degree δ_{out} is shaded in blue.

as the underlying primitive – for technical reasons the definition slightly differs from the one which we introduced in Definition 41 in Chapter 5 to prove adaptive security of the CGKA protocol TTKEM in the random oracle model. We will establish lower bounds for this version of public-key GSD analogously to the bounds for secret-key GSD, which will serve as a basis for the lower bound on TreeKEM in Section 7.7.

7.6.1 Lower Bounds for GSD

In many applications one considers games where the adversary's queries are restricted to certain graph structures, e.g., paths, "in-trees" (i.e. rooted trees directed from the leaves to the root), or low-depth graphs. These restrictions depend on the protocol under consideration and often allow to construct stronger reductions.

Interesting upper bounds are known for specific settings for (oblivious) black-box reductions R proving adaptive GSD security based on IND-CPA security (short, GSD reductions). Our results now allow us to prove lower bounds on Λ for GSD with various restrictions (which cover similar settings as known upper bounds). Note that our lower bounds are stronger and more widely applicable the more restrictions they can handle.

Definition 67 (Black-box and straight-line GSD reduction)**.** R is a *black-box* GSD reduction if for every secret-key encryption scheme $\mathsf{SKE} = (\mathsf{Enc}, \mathsf{Dec})$ and every adversary A that wins the GSD game played on SKE, R breaks SKE. Moreover, if A is an (t, ϵ) GSD adversary and R (t', ϵ')-breaks SKE (where t' and ϵ' are functions of t and ϵ) then the loss in security is defined to be $(t'\epsilon)/(t\epsilon')$. A black-box GSD reduction R is *straight-line* if it, additionally, does not rewind A.

The following definition mirrors the obliviousness of Pebblers in the context of the Builder-Pebbler Game (cf. Definition 60).

Definition 68 (Oblivious GSD reduction)**.** A straight-line GSD reduction R (Definition 67) is *oblivious* if it commits to a non-trivial vertex cover of all inconsistent edges at the beginning of the game.

In all our bounds we require the reduction to assign keys to nodes at the beginning of the game.

Definition 69 (Key-committing GSD reduction)**.** A black-box GSD reduction R is *key-commiting* if it commits to an assignment of keys to all nodes at the beginning of the game.

This is due to the fact that Pebblers in the Builder-Pebbler Game commit to whether an edge is pebbled or not as soon as they respond to the query. Without this requirement, this is not true for reductions in the GSD game, since they could potentially respond to a query and decide later if that edge is consistent or inconsistent by choosing the key for the target accordingly (as long as this node does not have an outgoing edge). However, this requirement should not be seen as a very limiting restriction, but we introduce it for ease of exposition, since there are several "work arounds" to this issue. 1) One could use an adversary that "fingerprints" the keys by querying the encryption of some message under each key before starting the rest of the query phase. This would entail adding the corresponding oracle to the GSD game, which seems reasonable in many (but not all) applications, since the keys are often

not created for their own sake, but to encrypt messages. 2) In case the adversary is not too restricted (which is application dependent), there is a generic fix where the adversary abuses the $\texttt{encrypt}$ oracle to achieve this fingerprinting by introducing a new node and querying the edges from every other node to this new node. This introduces only a slight loss in the number N of nodes.

Both of these approaches work, but would make the proof more complicated: recall that the challenge node must be a sink, so neither of the two fixes can be applied to it. We can still fix all other nodes (which is sufficient), thereby giving away the challenge node right at the start of the game. But this can only increase the reduction's advantage by a factor N, since it could also simply guess the challenge node. Since we are only interested in super-polynomial losses in this work, this would not affect the results. But for the sake of clarity we refrain from applying this workaround and simply keep this mild condition on the GSD reductions. Looking ahead, we will see that some protocols are based on a public key version of GSD (cf. 7.6.2) rather than the secret key version we consider here. In such cases the public keys are known to the adversary and commit the reduction to the corresponding secret keys and thus no assumption or extra fix are required.

We now give a general lemma that allows to turn lower bounds for the Builder-Pebbler Game into lower bounds for the GSD game.

Lemma 34 (Coupling lemma for GSD)**.** *Let $\mathcal{G}$ be a family of DAGs and X a cut function. Let* B *be an oblivious Builder in the $(N, \mathcal{G})$-Builder-Pebbler Game with winning condition X. Then there exists*

1. *an ideal SKE scheme* $\Pi = (\mathsf{Enc}, \mathsf{Dec})$
2. *a GSD adversary* A *in* PSPACE

such that for any key-committing straight-line *GSD reduction* R *there exists a Pebbler* P *such that the advantage $1/\Lambda$ of* R *is at most the advantage π of* P *against* B *(up to an additive term* $\mathrm{poly}(N)/2^{\Omega(N)}$*). Moreover, if* R *is oblivious then so is* P*.*

Proof. We first construct $\Pi = (\mathsf{Enc}, \mathsf{Dec})$: We will pick Enc to be a random expanding function (which is injective with overwhelming probability). More precisely, assuming (for simplicity) the key $\mathtt{k}$, the message $\mathtt{m}$ and the randomness r are all λ-bit long, $\mathsf{Enc}(\mathtt{k}, \mathtt{m}; r)$ maps to a random ciphertext of length, say, 6λ with $\lambda = \Theta(N)$. Dec is simulated accordingly to be always consistent with Enc.

We now define a map ϕ from GSD adversaries and reductions to Builder-Pebbler Game Builders and Pebblers:

- The number N of nodes in the Builder-Pebbler Game corresponds to the number N of keys in the GSD game.
- An encryption query $(\texttt{encrypt}, v_i, v_j)$ maps to an edge query (i, j) in the Builder-Pebbler Game.
- A response to a query $(\texttt{encrypt}, v_i, v_j)$ is mapped to "no pebble" if it consists of a valid encryption of $\mathtt{k}_j$ under the key $\mathtt{k}_i$, and to "pebble" otherwise. (Note that this is always well-defined for key-committing GSD reductions.)

- A corruption query $(\texttt{corrupt}, v_i)$ is ignored in the Builder-Pebbler Game.
- The challenge query $(\texttt{challenge}, v_t)$ is mapped to the challenge node t.

Let A $\in$ PSPACE be the following preimage of B under ϕ: A performs the same encryption queries as B and selects its GSD challenge node as the challenge node chosen by B. It then corrupts all nodes not in the challenge graph G^t. If there is an inconsistency (i.e. a pebble) in $G \setminus G^t$, A aborts and outputs 0. Finally, it uses its computational power to decrypt all the received ciphertexts and determines the resulting pebbling configuration $\mathcal{P}$ on G^t. If $\mathcal{P}$ is in the cut defined by the frontier $X(G^t)$, A outputs 0, otherwise it outputs 1. Clearly, A wins the GSD game against Π with probability 1. We will now show that the advantage of R in using the GSD-adversary A to break the IND-CPA security of Π is at most the advantage of P $= \phi$(R) against B (up to a negligible additive term).

Note that since Enc is a random function, the GSD game is entirely independent of the challenge bit b until the tuple $(\mathtt{k}, \mathtt{m}_b, r)$ such that $\mathtt{c}^* = \mathsf{Enc}(\mathtt{k}, \mathtt{m}_b; r)$ (where $\mathtt{c}^*$ is the challenge ciphertext) is queried to Enc. Since R is PPT, the probability of R doing this is at most poly$(N)/2^{\Omega(N)}$. Accordingly, to gain a larger advantage, R must send $\mathtt{c}^*$ to A as response to some edge query. Since B $= \phi$(A) is oblivious, the behaviour of A does not depend on $\mathtt{c}^*$ (and thus not on b) during the entire query phase. This means that the statistical distance of A induced by $b = 0$ and $b = 1$ is

$$\sum_{(\mathcal{P}_i, \mathcal{P}_j) \in \mathcal{P}^{G^t}} p_{i,j} |\Pr[\mathsf{A}(\mathcal{P}_i) \to 1] - \Pr[\mathsf{A}(\mathcal{P}_j) \to 1]|$$

where $p_{i,j}$ is the probability that the query phase results in the configuration $\mathcal{P}_i$ or $\mathcal{P}_j$ depending on $\mathtt{c}^*$. More formally, for an edge $(\mathcal{P}_i, \mathcal{P}_j)$ in the configuration graph $\mathcal{P}^{G^t}$, let $\mathcal{P}^c_{ij}$ be the "configuration" that is equal to $\mathcal{P}_i$ if $\mathtt{c}^*$ represents a consistent encryption edge (i.e. is not a pebble) and equal to $\mathcal{P}_j$ if $\mathtt{c}^*$ is inconsistent (i.e. a pebble). Then we define $p_{i,j}$ as the probability of the query phase resulting in $\mathcal{P}^c_{ij}$. Clearly, we have $|\Pr \mathsf{A}(\mathcal{P}_1) \to 1 - \Pr \mathsf{A}(\mathcal{P}_2) \to 1| = 0$ for any edge $(\mathcal{P}_1, \mathcal{P}_2)$ where $\mathcal{P}_1 \notin X(G^t)$ and 1 otherwise. The statistical distance of A induced by b is thus bounded by the probability of the querying phase ending up in a configuration in $X(G^t)$ (if $\mathtt{c}^*$ is considered not a pebble for this argument). This is exactly the advantage of Pebbler P $= \phi$(R) in the Builder-Pebbler Game against B. By data processing inequality, this also means that the advantage of R is bounded from above by the same quantity.

For the final statement of the lemma, note that ϕ maps oblivious GSD reductions to oblivious Pebblers. □

The following lower bounds on GSD now easily follow from Lemma 34 and the theorems in the previous section (Theorems 25, 27, 28 and 30, resp.).

Corollary 6 (Lower bound for GSD on paths, oblivious reductions)**.** *Let N be the number of users in the GSD game. Then any* key-committing oblivious *reduction proving adaptive GSD-security restricted to* paths *based on the IND-CPA security of the underlying encryption scheme loses at least a factor*

$$\Lambda \geq N^{\log(N)/8 - \log(\log(N))}.$$

Corollary 7 (Lower bound for GSD on binary trees, oblivious reductions)**.** *Let N be the number of users in the GSD game. Any* key-committing oblivious *reduction proving adaptive*

GSD-security restricted to rooted binary in-trees *based on the IND-CPA security of the underlying encryption scheme loses at least a factor*

$$\Lambda \geq N^{\log(N)-\log(\log(N))}.$$

For adversaries which are allowed to query any acyclic graph structure on N vertices and choose an arbitrary challenge, Theorem 28 gives the following result.

Corollary 8 (Lower bound for GSD on arbitrary DAGs, oblivious reductions)**.** *Any* key-committing oblivious *reduction proving adaptive security of unrestricted GSD with N users based on the IND-CPA security of the underlying encryption scheme loses at least a factor*

$$\Lambda \geq 2^{2(\sqrt{N/(e^3)}-1)}.$$

Finally, we can extend our results on paths to non-oblivious reductions. However, in this case our adversary needs to be able to query outside of the challenge graph and thus the restrictions we are able to handle outside of the challenge graph are not arbitrary as in Corollary 6.

Corollary 9 (Lower bound for GSD on trees, straight-line reductions)**.** *Let N be the number of users in the GSD game. Any* key-committing straight-line *reduction proving adaptive GSD restricted to* trees *based on the IND-CPA security of the underlying encryption scheme loses at least a factor*

$$\Lambda \geq N^{\log(N)/8}.$$

Even if the adversary is restricted to querying graphs with outdegree 2, the reduction loses at least a factor

$$\Lambda \geq N^{\log(N)/8-\log(\log(N))/4}.$$

7.6.2 Public-Key GSD

In this section we define the public-key analogue of GSD, where PKE is used instead of SKE as the underlying primitive. Such a public-key version of GSD was already introduced in Definition 41 in Chapter 5 to analyze the security of continuous group key agreement protocols like TreeKEM [BBR18, BBM+20]. We define a slightly different notion of public-key GSD in Definition 70 and then extend the lower bounds that we established for GSD in Corollaries 6 to 9 to public-key GSD (Corollaries 10 to 13). Corollary 13 will be used later in Section 7.7 to show a lower bound for TreeKEM. But first we highlight some important differences in the modelling of the game for public-key and symmetric-key GSD.

Public-key GSD vs. (symmetric-key) GSD. The natural way to adapt the notion of key-graph to public-key GSD, played with a PKE $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, is as follows:

- a vertex v is associated with a pair of public and secret keys $(\mathtt{pk}_v, \mathtt{sk}_v)$; and
- an edge (u, v) encodes the ciphertext $\mathsf{Enc}(\mathtt{pk}_u, \mathtt{sk}_v)$.

However, compared to symmetric-key GSD, there is a small subtlety with respect to the challenge node. In particular, the adversary must never learn the public key associated to the challenge node, since it could otherwise trivially distinguish the corresponding secret key from a random one. So instead one can think of the challenge node being associated only with a

secret key. (In fact, in applications like TreeKEM, the root is not associated with a PKE key pair, but with an SKE key or a seed used as input for a KDF.) We recommend the reader think of all secret keys in this game as randomness for the key generation algorithm. Then, secret keys of PKE key pairs can be thought of as secret keys for SKEs as long as the public key remains hidden.

In contrast to symmetric-key GSD, in public-key GSD there needs to be a mechanism for the adversary to learn the public keys, even if they are not all created at the beginning of the game. In Definition 41 in Chapter 5 this issue was solved by sending public keys of the source nodes along with encryptions. This also ensured that the adversary would never learn the public key of the challenge, since the challenge must be a sink, and thus can be thought of as being associated to a secret key. We will slightly change this mechanism and introduce a new oracle for this, `reveal`, which allows the adversary to retrieve the public keys of nodes. Using this oracle the adversary can learn the public keys of nodes that are currently sinks (but will not be sinks at a later stage) and thus commit the reduction to placing a pebble immediately after an edge query. Clearly, the adversary must not call this oracle on the challenge as discussed above.

This change is purely for technical reasons. We argue that the mechanism through which the adversary learns the public keys does not matter too much for the applications, since any reduction that relies on public keys remaining secret (even for a limited time) is probably not very meaningful.

Definition 70 (Public-key GSD)**.** Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme with secret key space $\mathcal{K}$ and message space $\mathcal{M}$ such that $\mathcal{K} \subseteq \mathcal{M}$. The *public-key GSD game* is a two-party game between a challenger C and an adversary A. On input an integer N, for each $i \in [1, N]$ the challenger C generates a key pair $(\mathtt{pk}_i, \mathtt{sk}_i)$ and initializes the *key-graph* $G = (\mathcal{V}, \mathcal{E}) := (\{v_1, \ldots, v_N\}, \emptyset)$, the set of corrupt users $\mathcal{C} = \emptyset$ and the set of revealed users $\mathcal{R} = \emptyset$. A can adaptively do the following queries:

- $(\mathtt{encrypt}, v_i, v_j)$: On input two nodes v_i and v_j, C returns an encryption $\mathtt{c} \leftarrow \mathsf{Enc}_{\mathtt{pk}_i}(\mathtt{sk}_j)$ of $\mathtt{sk}_j$ under $\mathtt{pk}_i$ and adds the directed edge (v_i, v_j) to $\mathcal{E}$.
- $(\mathtt{corrupt}, v_i)$: On input a node v_i, C returns $\mathtt{sk}_i$ and adds v_i to $\mathcal{C}$.
- $(\mathtt{reveal}, v_i)$: On input a node v_i, C returns $\mathtt{pk}_i$ and adds v_i to $\mathcal{R}$.
- $(\mathtt{challenge}, v_i)$, single access: On input a challenge node v_i, C samples $b \leftarrow \{0, 1\}$ uniformly at random and returns $\mathtt{sk}_i$ if $b = 0$, otherwise it returns a new secret key generated by Gen using a new independent uniformly random seed. In the context of GSD we denote the *challenge graph* as the graph induced by all nodes from which the challenge node v_i is reachable. We require that none of the nodes in the challenge graph are in $\mathcal{C}$, that G is acyclic and that the challenge node v_i is a sink and not in $\mathcal{R}$.

Finally, A outputs a bit b' and it *wins* the game if $b' = b$. We call the encryption scheme (t, ϵ)-adaptive GSD-secure if for any adversary A running in time t the distinguishing advantage is at most ϵ.

We now give a general lemma that is the analogue of Lemma 34, i.e., it allows to turn lower bounds for the Builder-Pebbler Game into lower bounds for the public-key GSD game. The definition of oblivious, straight-line and black-box reductions for public-key GSD can be defined similarly to (symmetric-key) GSD (i.e., Definitions 67 and 68).

Lemma 35 (Coupling lemma for public-key GSD)**.** *Let $\mathcal{G}$ be a family of DAGs and X a cut function. Let* B *be an oblivious Builder in the $(N, \mathcal{G})$-Builder-Pebbler Game with winning condition X. Then there exists*

1. *an ideal PKE scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$
2. *a public-key GSD adversary* A *in* PSPACE

such that for any key-committing straight-line *reduction* R *for public-key GSD there exists a Pebbler* P *such that the advantage* $1/\Lambda$ *of* R *is at most the advantage of* P *against* B *(up to an additive term* $\mathrm{poly}(N)/2^{\Omega(N)}$*). Moreover, if* R *is oblivious then so is* P*.*

Proof (Sketch). The proof of this lemma is similar to that of Lemma 34 except that we use an ideal PKE scheme as defined in [GMR01] in place of an ideal SKE scheme. Therefore, we only highlight the difference in the ideal scheme here. The key-generation algorithm Gen in Π is defined to be a random expanding function from λ bits to 2λ bits. The encryption function Enc is then defined similar to that in Lemma 34 except that the domain and co-domain are adjusted to be 4λ and 8λ to accommodate the public-key. Dec is defined to be consistent with Enc. □

The following lower bounds on public-key GSD now follow from Lemma 35 and the Theorems 25, 27, 28 and 30, resp., in Section 7.5. These are analogous to Corollaries 6 to 9 for GSD. Note that oblivious reductions for public-key GSD can be defined similar to Definition 60. As mentioned, we do not need to assume that the reduction is key-committing, since the public keys sent as response to the queries act as a fingerprint on the private key.

Corollary 10 (Lower Bound for Public-Key GSD on Paths, Oblivious Reductions)**.** *Let N be the number of users in the public-key GSD game. Then any* oblivious *reduction proving adaptive security for public-key GSD restricted to* paths *based on the IND-CPA security of the underlying encryption scheme loses at least a factor*

$$\Lambda \geq N^{\log(N)/8 - \log(\log(N))}.$$

Corollary 11 (Lower bound for public-key GSD on binary trees, oblivious reductions)**.** *Let N be the number of users in the public-key GSD game. Any* oblivious *reduction proving adaptive security for public-key GSD restricted to* rooted binary in-trees *based on the IND-CPA security of the underlying encryption scheme loses at least a factor*

$$\Lambda \geq N^{\log(N) - \log(\log(N))}.$$

Corollary 12 (Lower bound for public-key GSD on arbitrary DAGs, oblivious reductions)**.** *Any* oblivious *reduction proving adaptive security for unrestricted public-key GSD with N users based on the IND-CPA security of the underlying encryption scheme loses at least a factor*

$$\Lambda \geq 2^{2(\sqrt{N/(e^3)} - 1)}.$$

Corollary 13 (Lower bound for public-key GSD on trees, straight-line reductions)**.** *Let N be the number of users in the public-key GSD game. Any* straight-line *reduction proving adaptive security for public-key GSD restricted to* trees *based on the IND-CPA security of the underlying encryption scheme loses at least a factor*

$$\Lambda \geq N^{\log(N)/8}.$$

Even if the adversary is restricted to querying graphs with outdegree 2, the reduction loses at least a factor

$$\Lambda \geq N^{\log(N)/8 - \log(\log(N))/4}.$$

7.7 Cryptographic Lower Bound II: Continuous Group Key Agreement

The main motivation behind the definition of GSD in [Pan07] was to analyze the security of a particular protocol for multicast encryption [FN94] called *logical key hierarchy* (LKH) [WGL00]. The recent constructions of (asynchronous) continuous group key agreement (CGKA) protocols like TreeKEM [BBR18, BBM+20] (and its variant Tainted TreeKEM from Chapter 5) can roughly be regarded to be the public-key analogue of LKH – the security of TreeKEM (and its variants) is, in fact, analyzed using the public-key GSD game (Definition 70).

Although abstracting TreeKEM as public-key GSD suffices for establishing upper bounds for loss in adaptive security, we have to be careful when it comes to establishing lower bounds. In particular, the lower bounds established for public-key GSD in Section 7.6.2 do not quite carry over to a lower bound for TreeKEM. The high level reason is that the key-graphs that result in its security analysis are more 'structured' than in the case of public-key GSD, and as a result the querying strategies that we deployed in the public-key GSD adversaries in Corollaries 6 to 9 cannot be used in this setting per se.

Nevertheless, we show in Section 7.7.2 that it is possible to 'emulate' some of these strategies: in particular, the Builder strategy from Theorem 30 can be emulated *within* a TreeKEM adversarial strategy. Consequently, we get that any *straight-line* reduction for TreeKEM (and its variants) must lose a factor that is super-polynomial in M, the number of users (Corollary 14). Rather surprisingly, we are unable to show any results for LKH as the security model for multicast encryption is rather weak compared to that for CGKA (see Section 7.10.4).

7.7.1 Definitions and Construction

For sake of space, we keep the discussion on CGKA and TreeKEM at an informal level sufficient to explain the lower bound, and refer the readers to [ACDT20] and Chapter 5 (where we introduced Tainted TreeKEM – short, TTKEM – a variant of TreeKEM) for a more formal treatment.

CGKA: Syntax

CGKA is a public-key, multi-user primitive which involves M users denoted $\mathsf{ID}_0, \ldots, \mathsf{ID}_{M-1}$. Any user ID_i can initialise a group $\mathcal{U}_0 \subseteq \{\mathsf{ID}_0, \ldots, \mathsf{ID}_{M-1}\}$ by sending protocol messages to all group members, from which each group member can compute a shared group key $\mathtt{K}_0$. To this aim, ID_i must know the (current) public key $\mathtt{pk}_j$ of each invitee $\mathsf{ID}_j \in \mathcal{U}_0$. Once a group has been created, CGKA allows any group member ID_i to *update* their key using $(\mathtt{update}, \mathsf{ID}_i)$. The role of the update operation is to allow the user to 'refresh' their state in case their previous state was leaked to an adversary and help achieve post-compromise security. This is accomplished by replacing their old public key with a new one, and then accordingly updating the group key. Any group member ID_i can use $(\mathtt{add}, \mathsf{ID}_i, \mathsf{ID}_j)$ to *add* a new user ID_j, and use $(\mathtt{remove}, \mathsf{ID}_i, \mathsf{ID}_j)$ to *remove* an already-existing member ID_j. Carrying out these group

operations results in the evolution of the group through *epochs* from $\mathcal{U}_0$ to $\mathcal{U}_Q$, Q being the total number of epochs, with the corresponding group keys $\mathtt{K}_0, \ldots, \mathtt{K}_Q$.[15]

Handling asynchronicity. The *group operations* – `update`, `add` and `remove` – require sending protocol messages to all members of the group. Since it is not assumed that the parties are online at the same time (asynchronicity), all protocol messages are instead exchanged via an untrusted *delivery server*. It is possible that the server receives conflicting requests (e.g., $(\mathtt{remove}, \mathsf{ID}_i, \mathsf{ID}_j)$ and $(\mathtt{remove}, \mathsf{ID}_j, \mathsf{ID}_i)$). This is handled in CGKA via the `confirm` operation: $(\mathtt{confirm}, \mathsf{ID}_i, \mathsf{a}_i)$ is used to confirm or reject ID_i's request for a group operation a_i (ID_i then proceeds to update their own local state accordingly). In case the group operation a_i is confirmed, the server delivers the message to the members and a member $\mathsf{ID}_j \neq \mathsf{ID}_i$ uses $(\mathtt{process}, \mathsf{ID}_j, \mathsf{a}_i)$ to process the group operation. These two operations – `confirm` and `process` – constitute the *delivery operations*. Although the delivery server can always prevent any communication taking place, it is required that the shared group key in the CGKA protocol – and thus the messages encrypted in the messaging system built upon it – remains private. For a formal security definition we refer to Definition 36 in Chapter 5.

TreeKEM: Protocol and Security

For simplicity, we consider a vanilla version of TreeKEM which suffices to explain our lower bound. In particular, we make the following simplifying assumptions to the protocol (version 9) described in [BBM+20]. (The assumptions will make more sense once the protocol is described.)

1. TreeKEM uses a *propose-commit* mechanism, where one member ID_i 'proposes' a group operation g_i and another member ID_j (potentially same as ID_i) 'commits' g_i by executing it and generating the corresponding protocol messages. Moreover, batches of proposed group operations (assuming they are mutually-consistent) can be committed by ID_j in one go. We assume that the member proposing a group operation g_i (i.e., ID_i above) *themself* generate the protocol messages for g_i and leave it to the delivery server to confirm or reject g_i. Moreover, we prohibit batching and limit members to only one group operation per proposal.

2. TreeKEM is built on top of a PKE scheme and its group-key-generation algorithm invokes the key-generation algorithm of this PKE scheme multiple times. TreeKEM optimises this process using so-called *hierarchical key-derivation*, where a member ID_i uses a hash-based key-derivation function (HKDF) to generate the key-pairs (of the underlying PKE) on 'its path to the root'. The motivation is to decrease the communication complexity. We, on the other hand, assume that all the keys are generated *independently* by ID_i. This will result in a simpler key-graph at the cost of a slightly less efficient protocol.

3. The TreeKEM protocol is based on *ratchet trees* and its structure evolves over time along with the structure of the group. We assume

    a) unlike in TreeKEM, a priori upper bound $M = 2^m$ on the number of users, and therefore the size of the group; and

[15] Note that if the operation at epoch q was an update operation, then $\mathcal{U}_{q+1} = \mathcal{U}_q$ but the group key does get updated.

b) that the leaf $i \in \{0,1\}^m$ in the ratchet tree is *reserved* for the user ID_i: in TreeKEM the position of a user is *not* fixed and when a user is added it is assigned the 'leftmost' free leaf in the current ratchet tree.

As a result of these assumptions, the ratchet tree for every epoch will be the *perfect* binary tree B_m and we can avoid having to deal with extending this tree.

The lower bound we establish for vanilla TreeKEM can be extended to the TreeKEM protocol described in [BBM+20] and Tainted TreeKEM (Chapter 5) without much difficulty. Henceforth, by TreeKEM we refer to the vanilla formulation.

Ratchet tree. The TreeKEM protocol is based on so-called *ratchet trees* and the structure of the ratchet tree evolves over the epochs along with the structure of the group. The key-graph corresponding to the ratchet tree for an epoch $q \in [0, Q]$ is a *perfect* binary tree B_m of depth m (see Figure 7.4). The vertices in B_m are classified into two: normal and 'blank'. A normal vertex is associated with a key-pair of the underlying PKE scheme, whereas a blank vertex is not – the exact role of these blank vertices will be explained along with the $\mathtt{add}$ operation later in the section. The vertices have the following semantics[16]:

- Members are associated with leaves: the leaf $i \in \{0,1\}^m$ of member $\mathsf{ID}_i \in \mathcal{U}_q$ is associated their key-pair and is therefore normal. All unoccupied leaves are blank.
- The root ε is associated with the current group key $\mathtt{K}_q$ and is therefore normal. In fact, the root is normal for *all* epochs.
- Each normal *internal* vertex $v \in \{0,1\}^{<m}$ is associated with an *independently* sampled key-pair $(\mathtt{pk}_v, \mathtt{sk}_v)$.

The edges in B_m encode the ciphertexts generated to enable the members to compute the group key $\mathtt{K}_q$. To this end, the following *invariant* is maintained throughout the epochs: each member $\mathsf{ID}_i \in \mathcal{U}_q$ only knows the secret keys corresponding to the vertices on its *path* P_i to the root

$$i = i\,[0, m-1] \to i\,[0, m-2] \to \ldots \to i\,[0,1] \to i\,[0,0] \to \varepsilon$$

where $i[0,b]$ denotes the prefix of i of length $b+1$. Therefore, the protocol messages designated to a member $\mathsf{ID}_i \in \mathcal{U}_q$ consists of the ciphertexts $\mathsf{Enc}_{\mathtt{pk}_u}(\mathtt{sk}_v)$ for every edge $(u,v) \in E(P_i)$ (see Figure 7.4.(a)). Since there are no keys corresponding to a blank vertex, the idea is to ignore these vertices when computing ciphertexts, i.e.:

- If a leaf i is blank no ciphertext is computed for i. The same holds for any blank internal vertex v whose ancestors are all blank.
- When an internal vertex v (with some normal ancestor) is blank, it is skipped and the ciphertext corresponding to the (unique) successor v' of v is computed – in case v' is also blank this rule is applied recursively until a normal node is encountered (recall that the root is guaranteed to be normal).

[16] To be precise, each vertex v should be accompanied by a superscript which specifies the epoch it belongs to: e.g., $v^{(q)}$ when v is from epoch q. However, to avoid the cluttering introduced by this notation, we simply assume that the epoch to which a vertex belongs is clear from the context. In case there are two vertices that need disambiguation, we use a prime in the exponent, with the prime usually referring to v in a later epoch: e.g., v and v' instead of $v^{(q)}$ and $v^{(q+1)}$.

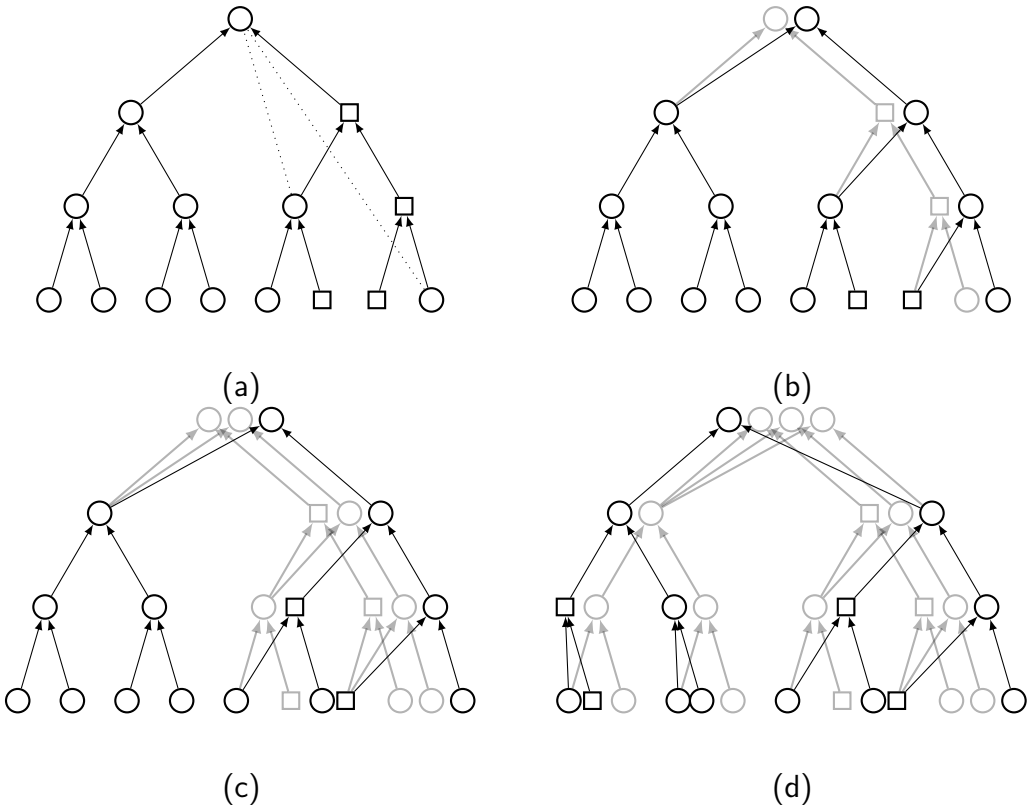


Figure 7.4: Ratchet tree and group operations in TreeKEM. (a) *The ratchet tree of the group in epoch* q. The blank (resp., normal) vertices are shown as squares (resp., circles). The group currently has six members $\mathcal{U}_q = \{\mathsf{ID}_0, \ldots, \mathsf{ID}_4, \mathsf{ID}_7\}$ assigned the leaves $000, \ldots, 100$ and 111 respectively. The leaves 101 and 110 are unpopulated and hence blank. The dotted edges represent the actual ciphertexts corresponding to paths involving blank vertices: the shorter (resp., longer) of the dotted edges represents the ciphertext $\mathsf{Enc}_{\mathrm{pk}_{10}}(\mathrm{K}_q)$ (resp., $\mathsf{Enc}_{\mathrm{pk}_{111}}(\mathrm{K}_q)$) which is encoded by the path $(10, 1) \to (1, \varepsilon)$ (resp, $(111, 11) \to (11, 1) \to (1, \varepsilon)$) . Note that there are no ciphertexts corresponding to the empty leaves 101 and 110. (b) *The key-graph after* ID_7 *updates themself.* A normal path P'_{111} from the leaf $111'$ to the (new) root ε' (and its co-path edges) replaces the old path P_{111} (and its co-path edges) which is shown in grey. This results in the new ratchet tree for epoch $q + 1$ as shown in black and the part of the key-graph in grey is outdated. (c) *The ratchet tree after* ID_5 *is added to the group by* ID_7. A blank path from the leaf $110'$ to the root (ε'') is added after which ID_7 updates themself (the two steps have been merged into one in the picture). Note that the node $10'$ is blanked in order to maintain the invariant: otherwise ID_7 would know a secret key associated to a node on the path from $\mathsf{ID}_4/\mathsf{ID}_5$ to the root. However further nodes on the path are normal as these also lie on the path from ID_7 to the root which lies outside its own path to the root. (d) *The ratchet tree after* ID_1 *is removed from the group by* ID_3. A blank path from the leaf $001'$ to the root (ε''') is added after which ID_3 updates themself (the two steps have been merged into one in the picture). The node $00'$ is blanked for the same reason as in (c).

For example, when all the internal vertices from a leaf $i \in \{0,1\}^m$ to the root are blank then there is a single ciphertext which is the encryption of the group key under the public key of ID_i.

Group operations. Since both `add` and `remove` are dependant on it, we start off with `update`. Looking ahead, `update` is the only *group* operation that our adversary (Algo-

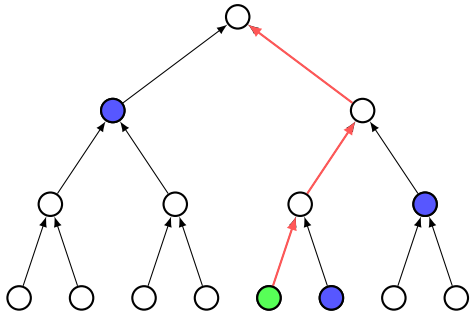


Figure 7.5: The path $P_{100} = 100 \to 10 \to 1 \to \varepsilon$ is highlighted in red. Its co-path vertices – 101, 11 and 0 – are also highlighted (in blue).

rithm 7.1) will employ (it also crucially uses the *delivery* operations though).

- *Update.* To update themself (see Figure 7.4.(b)), a user $\mathsf{ID}_i \in \mathcal{U}_q$ first generates a fresh path P'_i

$$i' \to i\,[0, m-2]' \to \ldots \to i\,[0,1]' \to i\,[0,0]' \to \varepsilon'$$

with *normal* vertices to the root. That is, for each vertex $v' \in V(P'_i)$ it generates a fresh key-pair $(\mathtt{pk}_{v'}, \mathtt{sk}_{v'})$ and for each edge $(u', v') \in E(P'_i)$ it generates a ciphertext $\mathsf{Enc}_{\mathtt{pk}_{u'}}(\mathtt{sk}_{v'})$. Next, to enable the other members in the group to process this update, it adds edges from all 'co-path' vertices (see Figure 7.5) of P_i to P'_i, i.e.,

$$\{v_0 \ldots v_{l-1}\bar{v}_l : v = v_0, \ldots, v_{l-1}v_l \in V(P_i)\},$$

where $\bar{v}_l$ is the bit complement. By the semantics described above, this means that for each co-path vertex u and its successor v' (which lies on P'_i and is hence normal), ID_i generates

  - ciphertext $\mathsf{Enc}_{\mathtt{pk}_u}(\mathtt{sk}_{v'})$ if u is normal; or
  - ciphertexts $\mathsf{Enc}_{\mathtt{pk}_w}(\mathtt{sk}_{v'})$ for each normal ancestor w of u such that all internal nodes on the path from w to v' are blank, in case u is blank.

Finally, it sends all the ciphertexts and the newly-generated public keys to the delivery server. In case the update is indeed confirmed, the group moves to the new epoch $q+1$ with the new ratchet tree B'_m obtained by replacing P_i and the edges incoming to it with P'_i and its co-path edges, while retaining the rest of B_m (see Figure 7.4.(b)).

- *Add and remove.* Suppose that a member $\mathsf{ID}_i \in \mathcal{U}_q$ wants to add a user ID_j to the group. According to the protocol specification ID_i does the following (see Figure 7.4):

  1. Perform an update operation 'on behalf of' ID_j but using *blank* internal vertices: i.e., ID_i adds a path P'_j that has a normal leaf j' (associated with the key-pair of ID_j) but with the rest of the vertices blank and then adds edges from the co-path vertices of P_j.
  2. Update themself.

  The reason to use blank vertices in Item 1 instead of normal vertices as, e.g., in Item 1 is to maintain the aforementioned invariant: if ID_i samples keys on behalf of ID_j then it would know secret keys corresponding to vertices other than the ones that lie on its own path to the root. It is worth mentioning that Items 1 and 2 have been separated above

only for the sake of exposition: in Figure 7.4 they have been clubbed together into one step. The steps for a member $\mathsf{ID}_i \in \mathcal{U}_q$ to remove another member $\mathsf{ID}_j \in \mathcal{U}_q$ is mostly similar to that in `add` (see Figure 7.4.(d)):

1. Perform an update operation 'on behalf of' ID_j but using *only blank* vertices: i.e., ID_i adds a path P'_j with only blank vertices and then adds edges from the co-path vertices of P_j.
2. Update themself.

Authentication. The (base) protocol described thus far only takes care of the privacy aspect of CGKA. It is thus secure in a restricted setting where the delivery server behaves honestly and the group is always in a consistent state (i.e., in an attack model where the adversary cannot make delivery queries). To deal with dishonest delivery servers, the full protocol employs authentication on top of the base protocol and, furthermore, each ratchet tree B_m is tagged with a

1. *tree hash*, a commitment to (the public part of) B_m; and
2. *transcript hash*, a commitment to the *history* of operations that led to B_m as in a blockchain.

Therefore, whenever a user ID_i commits a group operation a and the group enters a new epoch, they have to – in addition to the protocol message of the base protocol for a – compute the tree hash of the new ratchet tree and generate the transcript hash using the previous transcript hash, the new tree hash and the particulars of a. Moreover, another member ID_j processes a only if the accompanying hashes are consistent with the ones stored locally as part of their own state. This enables group members to ensure that they agree on the *public* cryptographic state of the group and guarantees that only users that are in *consistent state* can communicate with each other.

Security. As a consequence of the structure of the group operations, the overall key-graph consists of a DAG of depth m and in-degree two (but the out-degree can depend on the number and order of group operations). The security game for TreeKEM can therefore be considered to be a special case of public-key GSD played on graphs of depth m and in-degree two. A security reduction for such a public-key GSD game was shown in Section 5.3.4 in Chapter 5 with a quasi-polynomial (in $M = 2^m$) loss of security, which translates to the following theorem for TreeKEM.

Theorem 31 (Theorem 16, restated for TreeKEM)**.** *If the underlying PKE scheme is IND-CPA secure and the hash function is collision resistant then TreeKEM is CGKA-secure with a loss in security of $O(M^2Q^{O(m)})$, where $M = 2^m$ is the number of users and Q is the number of queries.*

7.7.2 Lower Bound for TreeKEM

Note that in comparison to public-key GSD, an adversary A trying to break TreeKEM does not have complete freedom over the structure of the key-graph since the edges added for group operations always form a path to a root (with additional edges from the co-path vertices) as described in Section 7.7.1. Furthermore, A cannot corrupt arbitrary nodes in the ratchet

tree but is restricted to corruption of users, which are associated with leaves in the tree. Therefore, the strategy of the public-key GSD adversary used in the lower bound, e.g., using Theorem 30 (via Lemma 35 and Corollary 13), does not directly carry over since it requires more fine-grained control on which edges are added and which nodes are corrupted. However, we show that the querying strategy there can still be emulated by a TreeKEM adversary through appropriate group and delivery operations. The adversarial query strategy is formally described in Algorithm 7.1, and below we provide an intuitive overview.

Overview of the query strategy. Recall that the Builder strategy B in Theorem 30 (with the overlap parameter $k = 1$) is to construct, level by level, a regular tree T_k with in-degree 1, out-degree (denoted here by) k and depth D. Let's denote the d-th vertex in the ℓ-th level of this tree by $v_{\ell,d}$. Our goal is to embed T_k into the TreeKEM key-graph (see Figure 7.6) with the root $v_{0,1}$ of T_k set as the user ID_0. Before describing the query strategy of our TreeKEM adversary A, we make two observations about the TreeKEM protocol (Section 7.7.1) and the attack model (Definition 36) that will help with the description:

1. The only way to increase the out-degree of a vertex v (leaf or internal) in the key-graph is *indirectly* by performing a group operation on a member ID_i such that v is a co-path vertex of the path P_i, or – in case of blank nodes – v is connected to a co-path vertex v' of P_i such that all nodes on the path from v to v' except v itself are blank (see Figure 7.4). We say that ID_i *extends* the vertex v. In case the group is in a consistent state – i.e., all members have processed all the group operations – then only vertices in the *current* ratchet tree can be extended. Therefore, if delivery operations are not available to an adversary then vertices 'in the past' cannot be extended.

2. The adversary can only *indirectly* corrupt an internal node v of the key-graph by (i) corrupting a user ID_i (on leaf i) that is an ancestor of v to obtain their internal state and (ii) using the (public) knowledge of protocol messages to decrypt the chain of ciphertexts from i to v. Moreover, only the current internal state of a user can be obtained via such corruptions. Even though a internal state might be part of the key-graph (in some user's view), these are inaccessible to a TreeKEM adversary due to the restriction placed by the CGKA attack model. In other words, the adversary is *not* permitted to corrupt 'in the past'.[17]

With these observations in mind, our approach to simulate B is as follows (the value of k and D in terms of M will be specified later in Corollary 14):

1. Let's assume the initial group was created by a user in the right half of B_m, i.e. all internal nodes on the path $0^m \to 0^{m-1}, \ldots, 0 \to \varepsilon$ from 0^m to the root are blank (see Figure 7.6.(a)). To embed the level 1, A updates a set $\mathcal{U}_{0,1}$ of k users such that each user $\mathsf{ID}_i \in \mathcal{U}_{0,1}$ extends 0^m – since we assumed the internal nodes from 0^m to the root to be blank, such users are guaranteed to exist. These level-0 updates result in a set of level-1 vertices $\mathcal{V}_1 := \{v_{1,0}, \ldots, v_{1,k-1}\}$, which A extends in Item 2 (see Figure 7.6.(b) and Figure 7.6.(c)).

[17] Note that adding this capability to the adversary strengthens the model. In fact the proof in [ACC+19] holds in this stronger model. This boils down to the fact that the public-key GSD game has no notion of time and it is possible to corrupt any node as long as it does not render the challenge trivial.

2. To embed level 2, for each level-1 vertex $v_{1,i} \in \mathcal{V}_1$ just added in Item 1, A fixes a set of k members $\mathcal{U}_{1,i}$ such that each member $\mathsf{ID}_j \in \mathcal{U}_{1,i}$ extends $v_{1,i}$. Then it *forks* the state of the group as follows:

   a) ID_j *processes* all messages ID_i processed so that they share the same group state,

   b) both ID_j and ID_i process the update of ID_i (which created node $v_{1,i}$), and

   c) ID_j extends $v_{1,i}$ by updating themself.

   At the end of these level-1 updates, A will have embedded a set of k^2 level-2 vertices $\{v_{2,0}, \ldots, v_{2,k^2-1}\}$ (see Figure 7.6.(d)).

3. As in B, A randomly selects a k-sized batch of level-2 vertices

$$\mathcal{V}_2 = \left\{v_{2,d_1^* k}, \ldots, v_{2,(d_1^*+1)k-1}\right\} \subset \{v_{2,0}, \ldots, v_{2,k^2-1}\},$$

   where $d_1^* \in [0, k-1]$, which will be extended to get the next level.

4. A repeats Items 2 and 3 above another $D-3$ times to complete the tree T_k: i.e., for $\ell \in [3, D]$, in the ℓ-th iteration

   a) vertices in $\mathcal{V}_{\ell-1}$ are extended by forking the group state to get level-ℓ vertices $\{v_{\ell,0}, \ldots, v_{\ell,k^2-1}\}$; and

   b) a random batch $\mathcal{V}_\ell = \left\{v_{\ell,d_{\ell-1}^* k}, \ldots, v_{\ell,(d_{\ell-1}^*+1)k-1}\right\} \subset \{v_{\ell,0}, \ldots, v_{\ell,k^2-1}\}$ is selected to be extended in the next iteration.

5. Finally, A chooses an update corresponding to a random level-D vertex in $v_{D,d_D^*} \in \mathcal{V}_D$ as the challenge epoch. Then, for each level ℓ, it *indirectly* (recall the second observation above) corrupts every level-ℓ vertex $v \in \mathcal{V}_\ell$ *bar* the one that lies in the challenge path

$$v_{0,d_0^*} = 0^m \to v_{1,d_1^*} \to \ldots \to v_{D,d_D^*}$$

   by corrupting the *member* which generated v. This is to make sure the reduction answers the edges not rooted in the final challenge graph honestly.

The query strategy is formally described in Algorithm 7.1 and an example of the resulting graph structure is shown in Figure 7.6. Note that A crucially exploits the process operation in Item 2 in order to force the group into an *inconsistent* state and get around the two issues we noted above, viz., inability to extend vertices in past ratchet trees or corrupt past states. Let's consider the first of the issues: if the level-0 updates made by $\mathcal{U}_{0,1}$ in Item 1 are *all* processed by *all* the members in the group – i.e., the group is in a *consistent* state – then all members in $\mathcal{U}_{1,j}$ would (by protocol specification) extend $v_{1,k}$, the level-1 vertex that was added in the *last* level-0 update. Instead, in Item 2, A makes only select members (i.e., $\mathcal{U}_{1,j}$) process select updates (i.e., level-0 update by $\mathsf{ID}_j \in \mathcal{U}_{0,1}$). This necessarily forks the state of the group, which means that the ratchet tree that a member *perceives* as current will be determined by the update they process. Consequently the vertices they extend will belong to what they perceive as the current ratchet tree (see Figure 7.6). However, this also means that members belonging to two different 'forks' of the group state will – due to the layer of authentication – no longer be able to communicate with each other (e.g., add each other). This is an issue though as A's query strategy does not require doing this anyway. Finally, note that forking the group state also solves the second issue: since the current ratchet tree of each fork is considered to be 'in the present' in the model, the members belonging to any fork can be corrupted (we still have to be careful not to render the challenge trivial though).

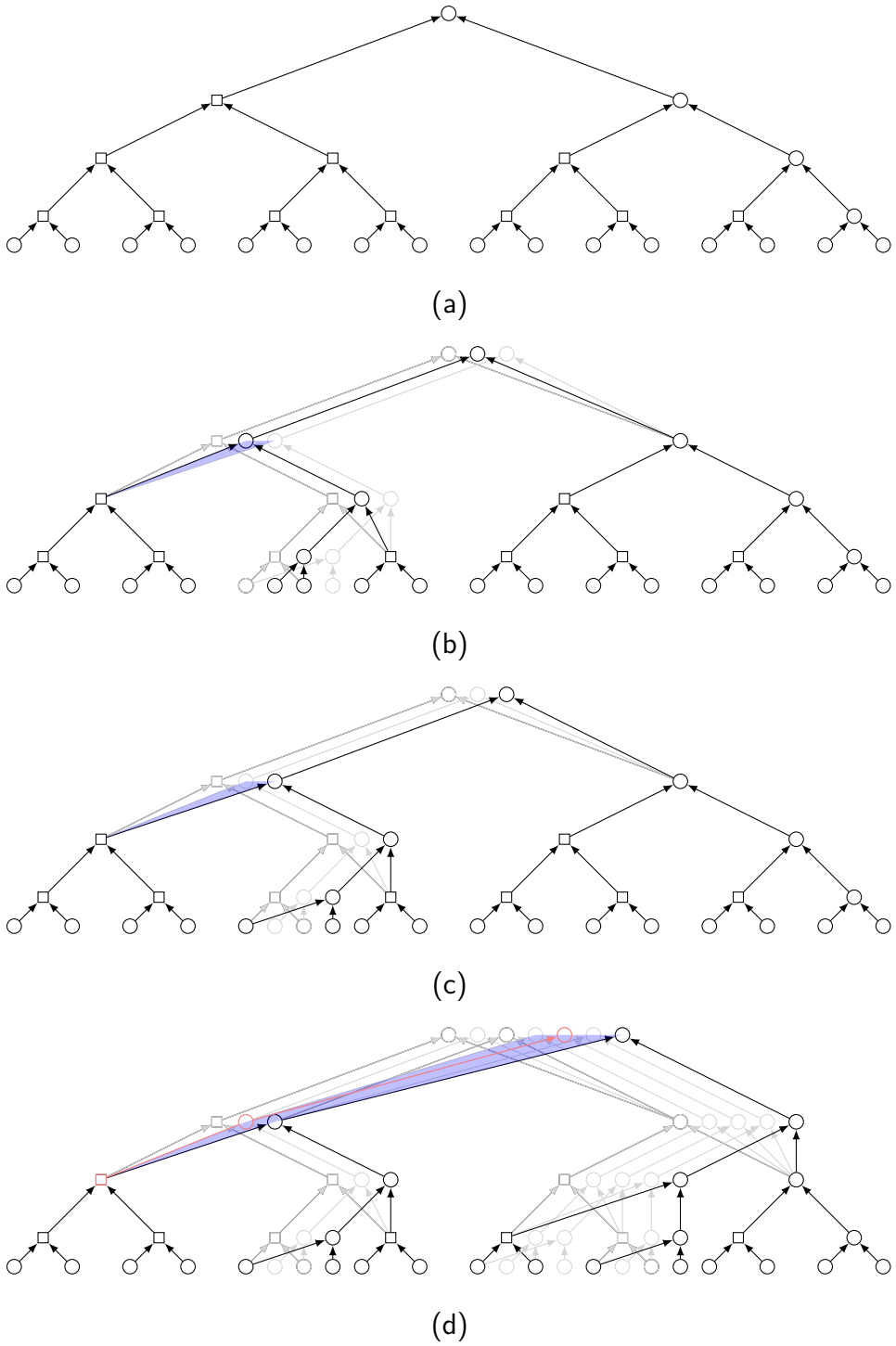


Figure 7.6: Embedding a regular tree of outdegree $k = 2$, depth $D = 2$ and overlap $k = 1$ in the TreeKEM key-graph. (a) *Initial ratchet tree.* The ratchet tree after the group was initialised by ID_{M-1} and all invitees process this initialisation. (b) and (c) *The key-graph after level-*0 *updates.* The key-graph after ID_4 on leaf 0100 and ID_5 on leaf 0101 update themself (*before* either operations are processed). These key-graphs have the same structure and the only difference is in what ID_4 and ID_5 perceive (viz., (b) and (c) respectively) as current ratchet tree after they process their own updates and the group state is forked. The first-level embedding is highlighted in blue (with $\mathcal{U}_{1,0} = \{\mathsf{ID}_4, \mathsf{ID}_5\}$). (d) *The key-graph after level-*1 *updates.* The point of view is of ID_{11} on leaf 1011. The embedding is complete (with $\mathcal{U}_{2,0} = \{\mathsf{ID}_8, \mathsf{ID}_9\}$ and $\mathcal{U}_{2,1} = \{\mathsf{ID}_{10}, \mathsf{ID}_{11}\}$): since the vertices 000 and 00 on the path $0000 \to 000 \to 00$ are blanked, the embedding actually is rooted at 0000 according to our convention for blank vertices from Section 7.7.1. The challenge path is highlighted in red.

Algorithm 7.1: Query strategy for the TreeKEM adversary A, parametrised by the number of users $M = 2^m$ and degree of embedding $k = 2^\delta$. Only details pertaining to the embedding of T_k has been included.

1 $(\texttt{initialise}, \mathsf{ID}_{M-1}, \{\mathsf{ID}_0, \ldots, \mathsf{ID}_{M-1}\})$ // ID_{M-1} fully populates the group, ID_i assigned leaf i
2 **for** *each* $i \in \{0,1\}^m$ **do**
3 $(\texttt{process}, 0, \mathsf{ID}_i)$ // All users process the group initialisation // $\mathcal{U}_{0,1}$ set as $\{\mathsf{ID}_{k^2}, \ldots, \mathsf{ID}_{k^2+k-1}\}$
for $d \in [0, k-1]$ **do** // Each user $\mathsf{ID}_d \in \mathcal{U}_{0,1}$...
4 $(\texttt{update}, \mathsf{ID}_{k^2+d})$: query $\texttt{update}_{0,d}$ // ...updates themself to extend root 0^m and generate level-1 vertices $\mathcal{V}_1 = \{v_{1,0}, \ldots, v_{1,k-1}\}$
5 Set $d_0^* := 0$ // Challenge path $v_{0,d_0^*} \to \ldots \to v_{D,d_D^*}$ initiated at the leaf 0^m
6 **for** $\ell \in [1, m-2\delta]$ **do** // For each higher level ℓ of T_k...
7 **for** $c \in [0, k-1]$ **do** // fork the group for each extendable $v_{\ell,d_{\ell-1}^* k+c} \in \mathcal{V}_\ell$
// Set $\mathcal{U}_{\ell,c} := \{\mathsf{ID}_{2^\ell k^2+ck}, \ldots, \mathsf{ID}_{2^\ell k^2+(c+1)k-1}\}$ and $\mathsf{ID}_i := \mathsf{ID}_{2^{\ell-1}k^2+d_{\ell-1}^* k+c}$
for *each* $d \in [0, k-1]$ **do** // For each user $\mathsf{ID}_j = \mathsf{ID}_{2^\ell k^2+ck+d} \in \mathcal{U}_{\ell,c}$...
8 **for** $l \in [0, \ell-2]$ **do** // ID_j processes all messages that...
9 $(\texttt{process}, \texttt{update}_{l,d_l^* k}, \mathsf{ID}_{2^\ell k^2+ck+d})$ // ...ID_i processed and both ID_i and ID_j end up in the same state
10 $(\texttt{process}, \texttt{update}_{\ell-1,d_{\ell-1}^* k+c}, \mathsf{ID}_{2^{\ell-1}k^2+d_{\ell-1}^* k+c})$ // Both ID_i and...
11 $(\texttt{process}, \texttt{update}_{\ell-1,d_{\ell-1}^* k+c}, \mathsf{ID}_{2^\ell k^2+ck+d})$ // ID_j process ID_i's update
12 $(\texttt{update}, \mathsf{ID}_{2^\ell k^2+ck+d})$: query $\texttt{update}_{\ell,ck+d}$ // ID_j extends $v_{\ell,d_{\ell-1}^* k+c}$ // Added $v_{\ell+1,ck}, \ldots, v_{\ell+1,(c+1)k-1}$
Sample $d_\ell^* \leftarrow [0, k-1]$ // Part of level ℓ to be extended in iteration $\ell+1$
13 $(\texttt{challenge}, \texttt{update}_{m-2\delta,d_{m-2\delta}^*})$ // Challenge epoch q^*
14 **for** $\ell \in [1, m-2\delta-1]$ **do** // For each (but extreme) level ℓ of T_k indirectly...
15 **for** $d \in [0, k-1] \setminus \{d_{\ell-1}^*\}$ **do** // ...corrupt the unextended level-ℓ vertices by...
16 $(\texttt{start-corrupt}, \mathsf{ID}_{2^\ell k^2+d_l^* k+d})$ // ...corrupting the user that generated it
17 $(\texttt{end-corrupt}, \mathsf{ID}_{2^\ell k^2+d_l^* k+d})$

The lower bound. In Lemma 36 we establish a tight coupling between the security game for TreeKEM and the Builder-Pebbler Game played on trees. Our lower bound for TreeKEM, Corollary 14, follows once the parameters are set appropriately.

Lemma 36 (Coupling lemma for TreeKEM)**.** *Let $\mathcal{G}$ be the family of DAGs of depth m and size N. Furthermore, let* B *and $X_{\mathcal{C}_m,1}$ be the Builder and the cut from Theorem 30. Then there exists*

1. *an ideal PKE scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$
2. *a CGKA adversary* A *in* PSPACE

such that for any straight-line *reduction* R *that proves CGKA security of TreeKEM for groups of size* $M = 2^m$ *based on the security of the underlying PKE scheme there exists a Pebbler* P *such that the advantage* $1/\Lambda$ *of* R *is at most the advantage of* P *against* B *(up to a negligible additive term* $\mathsf{poly}(M)/2^{\Omega(M)}$*).*

Proof (Sketch). The proof proceeds similar to Lemma 35. The ideal PKE is defined exactly as in Lemma 35 (with $\lambda = \Theta(M)$) and the query strategy of the adversary A is defined in Algorithm 7.1. The map ϕ from the CGKA game for TreeKEM to Builder-Pebbler Game can be carried out in two steps: in the first step we map the CGKA game for TreeKEM to the public-key GSD game by simply following the protocol description, and in the second step we use the map from Lemma 35 to end up with a Builder-Pebbler Game. Since Algorithm 7.1 embeds a k-regular tree T_k of depth m in the TreeKEM key-graph G, it follows that the corresponding Builder B obtained by mapping Algorithm 7.1 using ϕ also ends up building a graph G such that T_k is embedded in it. The proof now follows by the observation that the bound we established for Lemma 35 can be extended to any graph that has T_k embedded in it and because the order in which the embedding is carried out in G is exactly as by the Builder in Lemma 35. □

Corollary 14 (Lower bound for TreeKEM)**.** *Let* $M = 2^m$ *be an upper bound on the number of users. Then any* straight-line *reduction proving CGKA security of TreeKEM based on the security of the underlying encryption scheme loses at least a factor*

$$\Lambda \geq M^{\Omega(\log(m))}.$$

Proof. Simply set $k = 1$, $k = M^{1/4}$ (denoted δ_{out} there) and $D = m/2$ in Equation (7.2) in the proof of Theorem 30 to arrive at an upper bound of $\pi \leq 1/M^{\mathsf{O}(\log(m))}$ on the Pebbler's advantage. This proves the claim. □

Remark 13. It is possible to slightly improve on the constant factors in the exponent in Corollary 14 by using a more frugal query strategy that uses `add` and `remove` operations instead of just `update`. However, there will still remain a significant asymptotic gap in the upper bound from Theorem 31 and our lower bound (see Table 7.1). It is unclear whether or not the techniques used in the lower bound can be extended to close this gap and therefore a resolution in either direction is an interesting open question.

7.8 Cryptographic Lower Bound III: Constrained Pseudorandom Function

In this section we use our combinatorial results for the Builder-Pebbler Game to prove that the constrained pseudorandom function (cPRF) [BW13, BGI14, KPTZ13] based on the GGM PRF [GGM84] cannot be proven adaptively-secure based on the security of the underlying pseudorandom generator (PRG) using a *straight-line* reduction. Our lower bound almost matches the best-known upper bound by Fuchsbauer et al. [FKPR14], see Section 3.4 in Chapter 3. For definition, construction and security assumption, we refer to Section 3.4.1.

7.8.1 Lower Bound for the GGM cPRF

To prove a lower bound for GGM, we use the combinatorial upper bound from Section 7.5.3 for non-oblivious Pebblers, restricted to the class of graphs with outdegree 2. The main

challenge here is that – in contrast to our Builder from Section 7.5.3 – the constrain queries of an adversary in the security game for prefix-constrained PRFs correspond to paths in an exponentially large binary tree (see Figure 7.7). But it's not only that the adversary has to follow a certain query pattern, but more importantly for each query (which corresponds to a path of up to n edges) it only receives a single evaluation (and this evaluation allows A to efficiently compute any evaluations for the entire subtree below it). While A might be able to use its unrestricted computational power to distinguish whether the answer to its query lies in the image of the PRG (for an appropriately chosen PRG), it is impossible to extract a pebbling configuration on the entire path given just the single evaluation. This is why we follow a different approach and instead of choosing a PRG with sparse output range construct a PRG from two random permutations, which allows A to invert the function and compare whether two queries were computed from the same seed. Similar to the Builder strategy in Section 7.5.3, our adversary A makes bunches of queries forming complete binary subtrees, threaded along the challenge path. However, these queries are now paths of length n such that their *prefixes* cover the binary subtrees, respectively. Accordingly, we then map these bunches of queries to a pebbling strategy on the corresponding binary subtrees, instead of mapping single edges to a pebble or no pebble, as we did in previous applications. Fortunately, the combinatorial bound from Section 7.5.3 still holds for Builders revealing such bunches of queries at once.

Lemma 37 (Coupling lemma for GGM cPRF)**.** *Let $\mathcal{G}$ be the family of trees of depth D, size $N = \mathrm{poly}(D)$, indegree 1, outdegree 2 and a single source; i.e. $\mathcal{G}$ denotes the set of $\mathrm{poly}(D)$-sized subtrees of the binary tree of depth D which include the root, where edges are directed from the root to the leaves. Furthermore, let* B *and $X_{\mathcal{C}_D,k}$ for $k = \log(D)/2$ be the Builder and the cut from Theorem 30. Then there exists*

1. *an information-theoretically secure length-doubling PRG scheme* PRG
2. *a cPRF adversary* A *in* PSPACE

such that for any straight-line *reduction* R *that proves cPRF security of the GGM construction for input length $D + 1$ based on the security of the underlying PRG scheme there exists a Pebbler* P *such that the advantage $1/\Lambda$ of* R *is at most the advantage of* P *against* B *(up to a negligible additive term $\mathrm{poly}(D)/2^{\Omega(D)}$).*

To prove this lemma, we will use the following construction of an information-theoretically secure PRG scheme (see Definition 11).

Lemma 38. *Let $\pi_0, \pi_1 : \{0,1\}^\lambda \to \{0,1\}^\lambda$ be two random permutations. Then* PRG $: \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ *defined by* $\mathsf{PRG}(x) := (\pi_0(x), \pi_1(x))$ *is a $\mathrm{poly}(\lambda)/2^{\lambda/2}$-secure length-doubling PRG.*

Proof. Since random permutations are indistinguishable from random functions using only polynomially many queries, we may consider the PRG as a concatenation of two $\mathrm{poly}(\lambda)/2^{\lambda/2}$-secure PRFs by a hybrid argument. Again by hybrid argument, the concatenation of two secure PRFs yields a PRF from $\{0,1\}^\lambda$ to $\{0,1\}^{2\lambda}$. The lemma follows, since length extending PRFs are PRGs. □

Having a construction of a PRG in place, we are now ready to prove Lemma 37.

Proof of Lemma 37. We pick the PRG from Lemma 38 for $\lambda = \Theta(D)$.

Analogously to the proof of Lemma 34 we define a map ϕ between the cPRF game and the Builder-Pebbler Game:

- For a constrain query by adversary A, $(\texttt{constrain}, x)$, we make a case distinction on the length l of x:
  - if $l = D + 1$, the Builder B extends the current tree in the natural way, ignoring k-sized blocks of trailing zeros in x and adding random nodes as needed. More formally, write $x = x^1||x^2||x^3 \in \{0,1\}^{l_1} \times \{0,1\}^{l_2} \times \{0\}^{l_3} \times \{0,1\}$ with $l_1, l_2, l_3 \geq 0$ and $k|l_3$, where x^1 is the longest prefix of x that has been queried so far. For each prefix x' of x with length between $l_1 + 1$ and $l_1 + l_2$, B chooses a uniformly random node (that is not associated to any prefix yet) and associates it to x'. Writing $x^2 = (x_1^2, x_2^2, \dots)$, it then queries the edges between the nodes associated with x^1 and $x^1||x_1^2$, between $x^1||x_1^2$ and $x^1||x_1^2||x_2^2$, etc.
  - if $l \leq D$, B ignores the query.
- For the challenge query $(\texttt{challenge}, x^*)$, proceed as for constrain queries to extend the tree. Choose the node associated to x^* as the challenge T.
- Pebbles are determined in the following way. Recall that the Builder from Theorem 30 always extends the tree in chunks of entire subtrees (and the queries comprising such a chunk can be sent at the same time). So we may restrict the definition of ϕ to preimages of such Builders. To determine which edges in such a subtree are pebbled, consider the responses y_i corresponding to the queries x_i in such a chunk. For each y_i invert π_0 repeatedly to obtain the seed associated to the i-th leaf in the subtree. Then for every node, bottom-up, if
  - the children are associated with seeds s_0, s_1, resp., check if $\pi_0^{-1}(s_0) = \pi_1^{-1}(s_1)$. If this is true, associate the node with this computed seed. Otherwise, consider both outgoing edges from this node as pebbled and set the seed of this node to $\perp$.
  - only the left (right) child is associated with a seed s, set the seed of this node to $\pi_0^{-1}(s)$ ($\pi_1^{-1}(s)$, resp.).
  - neither of the children is associated with a seed, set the seed of the current node to $\perp$.

  For the root of the subtree, which already has a seed s (or $\perp$) associated to it, check if s is consistent with its children; if not, update to $\perp$ and pebble both outgoing edges.

Let A be the preimage under ϕ of B from Theorem 30 as follows: A first queries cPRF evaluations for $\{0,1\}^{2k}||0^{D-2k+1}$ in reverse order (i.e. starting from $1^{2k}||0^{D-2k+1}$)[18] – this is in analogy to the first $2k$ rounds of B (see Figure 7.7). Then it proceeds in $[1, D/k - 2]$ rounds, where in round $j \in [1, D/k - 2]$ it first samples $x_j^* \in \{0,1\}^k$ and then makes 2^{2k} queries $x_1^*|| \dots ||x_j^*||\{0,1\}^{2k}||0^{D-(j+2)k+1}$ in reverse order, starting with $x_1^*|| \dots ||x_j^*||1^{2k}||0^{D-(j+2)k+1}$.

[18] This is for technical reasons: We defined the mapping ϕ to ignore k-blocks of trailing zeros in order to associate queries $x||0^{D-2k+1}$ to (non-disjoint) paths of length $2k$. To this aim ϕ prolongs the longest already existing subpath associated to some prefix x' of x. If A now starts querying the string 0^{D+1}, this query would simply be ignored. On the other hand, if there was a preceding query $0^{2k-1}||1||0^{D-2k+1}$, then the query 0^{D+1} is mapped to an edge extending the path associated with the prefix 0^{2k-1}.

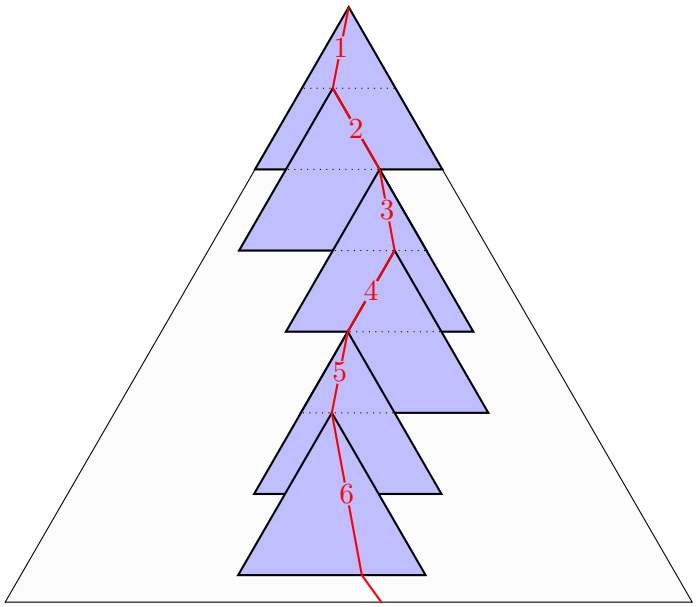


Figure 7.7: A schematic diagram showing the adversarial query strategy for the GGM cPRF in Lemma 37. The outer (grey) triangle represents the perfect binary tree of depth $D = 7k + 1$ (also see Figure 7.3) representing the GGM PRF. The internal (blue) triangles represent perfect binary trees of depth $2k$ with the j-th triangle representing the 2^{2k} queries $x_1^*||\ldots||x_j^*||\{0,1\}^{2k}||0^{D-(j+2)k+1}$. The challenge x^* is highlighted (in red) with the label j indicating the string x_j^*.

Next, A samples a challenge $x^* = (x_1^*, \ldots, x_D^*, 1)$ in $x_1^*||\ldots||x_{D/k-2}^*||\{0,1\}^{2k}||1$ uniformly at random. Furthermore, it makes constrain queries for all prefixes $(x_1^*, \ldots, x_{j-1}^*, \bar{x}_j)$ for $j \in [1, D]$. If the answers to the prefixes are not consistent with the previous cPRF queries, then A aborts and outputs 0. Otherwise, A uses its unrestricted computational power to compute the mapping ϕ from the reduction's answers to its queries to a pebbling configuration on the subtree. Note that due to the previous check, there must not be any pebbles on edges rooted at nodes outside the challenge path. A now considers the pebbling configuration induced on the challenge path. If this pebbling configuration lies in the cut defined by $X_{\mathcal{C}_D,k}$, the adversary A outputs 0, otherwise 1.

Clearly, A wins the cPRF game with probability 1. Now, let R be an arbitrary straight-line reduction. First, note that the probability that R queries PRG on the challenge seed is negligibly small (poly$(D)/2^{\Omega(D)}$). Assuming this does not happen, R can only gain a bigger advantage if it embeds its PRG challenge when interacting with A and manages to hit a pebbling configuration in the cut, i.e. such that depending on the challenge being real or random the pebbling configuration which A extracts lies either in the cut set or not. Note that choosing a value in the tree at random instead of applying PRG to the correct output is equivalent (w.r.t. A's behavior) to responding to the respective queries inconsistently and will thus yield a pebble with overwhelming probability. Furthermore, the consistency check after the constrain queries ensures that R may only place pebbles on edges rooted in the challenge graph and can only embed its challenge in the challenge graph. Similar to the proof in Lemma 34, one can see that R maps (under ϕ) to a Pebbler in the Builder-Pebbler Game which has at least the same advantage of achieving such a configuration. □

Using the above lemma, the following corollary now easily follows from Theorem 30.

Corollary 15 (Lower bound for GGM)**.** *Let n be the input length of the GGM cPRF scheme. Then any straight-line reduction proving cPRF security of the GGM construction based on the security of the underlying PRG scheme loses at least a factor*

$$\Lambda \geq n^{\log(n)/2-\log(\log(n))/2}.$$

7.9 Cryptographic Lower Bound IV: Proxy Re-encryption

As an application of our results on node Pebblers, we consider our results from Chapter 4 on adaptively secure proxy re-encryption (PRE). In particular, we identified two natural security properties – indistinguishability of ciphertexts and δ-weak key privacy – which allow to prove adaptive CPA-security via an (oblivious) black-box reduction. The notion of δ-weak key privacy clearly translates to node pebbling and a relation to the more general edge pebbling is not clear. The results from this chapter now allow us to prove lower bounds on the security loss involved by *any* black-box reduction that proves adaptive CPA security of a PRE scheme based on these two basic security properties.

Remark 14. Also other applications of the Piecewise-Guessing framework, such as Secret Sharing [JKK+17a] and Yao's Garbled Circuit (Chapter 6), as well as the recent application to ABE by Kowalczyk and Wee [KW19] use node-pebbling reductions. However, in all three of these applications of the framework, the graph structure is known to the reduction in the beginning of the game, which allows for some compression of the representation of the pebbling configurations. (However, we will show lower bounds for Yao's Garbled Circuit in Chapter 8.)

Recall, a PRE scheme is a public-key encryption scheme that allows the holder of a key $\mathtt{pk}$ to derive a re-encryption key (short, rekey) $\mathtt{rk}$ for any other key $\mathtt{pk}'$ [BBS98b]. This rekey lets anyone transform ciphertexts under $\mathtt{pk}$ into ciphertexts under $\mathtt{pk}'$ without having to know the underlying message. We say that a PRE is *unidirectional* if $\mathtt{rk}$ does not allow transformations from $\mathtt{pk}'$ to $\mathtt{pk}$ [AFGH05]. Moreover if ciphertext c' for $\mathtt{pk}'$ that was derived from a ciphertext c for $\mathtt{pk}$, can be further transformed to another ciphertext c'' corresponding to public key $\mathtt{pk}''$ using a rekey $\mathtt{rk}'$, the PRE is said to allow two "hops". A PRE that allows multiple hops, i.e. a multi-hop PRE, can be defined analogously.

A more formal definition of multi-hop, unidirectional PRE, to which we apply our lower bounds, is given in Section 4.2.

We consider the CPA-security from Chapter 4 as defined in Game 4.3. In the security game an adversary first receives the public keys of all users and then can adaptively do the following queries: It can corrupt a party and receive its secret key, it can query for rekeys between two users or for a re-encryption of a ciphertext encrypted under the public key of one user to an encryption of the same plaintext under the public key of another user, and, only once, it can issue a challenge query where it chooses a challenge user, two messages m_0, m_1 as well as a level and receives an encryption of m_b to the chosen levelunder the challenge user's public key. The adversary's goal is to guess the bit b.

Consider the graph structure on the set of users which is defined throughout the game as follows: Whenever the adversary queries a rekey or a reencryption of some ciphertext from

user i to user j, this is represented as an edge from j to i (note, for CPA-security we do not distinguish between rekey and reencryption queries).[19]

To avoid trivial wins, we need to restrict the adversary so that it can not simply reencrypt the challenge ciphertext to a corrupted party and then use the known secret key to decrypt. Thus, for CPA-security[20], the adversary is not allowed to query any paths of rekey or reencryption queries from the challenge user to a corrupted user. Considering the query graph, this corresponds to the requirement that the challenge node is not reachable from any corrupt node.

In Chapter 4 we reduced the CPA security of a PRE scheme to the following two basic security properties which are naturally satisfied by the popular constructions we analyzed. The first basic security property is *indistinguishability of ciphertexts*, as defined for public-key encryption in [GM82], but on all levels; see Definition 22. The second security property is δ-weak key privacy, which says that a set of δ re-encryption keys $\mathrm{rk}_{0,i}$ from a given source key $(\mathrm{pk}_0, \mathrm{sk}_0)$ to δ given target public keys pk_i, where $i \in [1, \delta]$, is indistinguishable from a set of δ rekeys which were generated from a freshly sampled source key pair $(\mathrm{pk}'_0, \mathrm{sk}'_0)$; see Definition 23.

7.9.1 Lower Bounds

In applications of PRE schemes it often makes sense to only consider security against restricted classes of adversaries where the recoding graph can only have a specific form, such as a path (e.g., in the application of key rotation) or binary trees (e.g., in a hierarchy of low depth). While for these cases quasi-polynomial upper bounds on the security loss involved when proving CPA-security of the PRE scheme based on IND-CPA security and δ-weak key privacy are known (see Chapter 4), our results allow us to prove quasi-polynomial lower bounds for all *oblivious* reductions, which basically means, that only the development of new techniques can lead to significally better reductions and hence stronger security guarantees.

Definition 71 (Oblivious PRE reduction)**.** A straight-line PRE reduction R is *oblivious* if it commits to a non-trivial vertex cover of all inconsistent edges (rekey queries) at the beginning of the game.

In all our bounds we require the reduction to assign keys to nodes at the beginning of the game.

Definition 72 (Key-committing PRE reduction)**.** A PRE reduction R is *key-committing* if it commits to an assignment of keys to all nodes at the beginning of the game.

Lemma 39 (Coupling lemma for PRE)**.** *Let $\mathcal{G}$ be a family of DAGs and X a cut function. Let* B *be an oblivious Builder in the $(N, \mathcal{G})$-Builder-Pebbler Game with winning condition X. Then there exists*

[19] In Chapter 4, edges were defined in a more natural way, opposite to here, which led to an *inverse* pebbling game where a node can be pebbled/unpebbled if all its children are pebbled. For the ease of presentation, we chose to define the query graph so that it fits our general framework and the usual reversible pebbling game. Analogously to the GSD game, corrupting a node allows the adversary to decrypt ciphertexts encrypted under the public key of any node which is reachable from it in the graph.

[20] In Chapter 4, we also considered the stronger and less restrictive notion of security under *honest re-encryption attack* (HRA) which was introduced in [Coh19] and distinguishes between (iterated) re-encryptions of the challenge ciphertext and unrelated ciphertexts. There we proved HRA-security for PRE schemes which satisfy one more basic property called *source-hiding*. Our lower bounds also hold for black-box reductions proving HRA-security of the scheme based on these three basic properties.

1. *an ideal PRE scheme* $\Pi = (\mathsf{S}, \mathsf{K}, \mathsf{RK}, \mathsf{E}, \mathsf{D}, \mathsf{RE})$
2. *a PRE adversary* A *in* PSPACE

such that for any key-committing straight-line *reduction* R *that proves adaptive PRE-CPA security of a PRE scheme based on the IND-CPA security of the underlying PKE scheme and the* δ*-weak key privacy there exists an oblivious Pebbler* P *such that the advantage* $1/\Lambda$ *of* R *is at most the advantage of* P *against* B *(up to an additive term* $\mathrm{poly}(N)/2^{\Omega(N)}$*). Moreover, if* R *is oblivious, then so is* P*.*

Proof. The proof is analogous to the one of Lemma 34, so we only point out the differences here. The ideal PRE scheme Π is defined as follows: we build on the ideal public-key encryption scheme from Section 7.6.2, from which ideal IND-CPA security follows. We now equip the PKE scheme with PRE capabilities by defining RK to respond with the output of a random function (with large enough co-domain, so that rekeys are sparsely distributed in the range of the function) under the query input. Upon re-encryption queries, the oracle 1) computes the secret and public keys that are consistent with the rekey and the ciphertext, 2.a) if the source key pair of the rekey coincides with the key pair associated with the ciphertext, it correctly decrypts the ciphertext and re-encrypts the message using the target public key of the rekey, 2.b) and otherwise (i.e., if the source key pair of the rekey and the key pair associated with the ciphertext do not match), it outputs a uniformly random string from the ciphertext space (i.e., co-domain of E). The scheme described is clearly correct, and one can show that it satisfies weak key privacy information-theoretically.

We now decribe the map ϕ that maps the parties in the PRE game to parties in a Builder-Pebbler Game:

- The number N of nodes in the Builder-Pebbler Game corresponds to the number N of keys in the PRE game.
- A rekey query $(\mathtt{rekey}, i, j)$ maps to an edge query (j, i) (sic!) in the Builder-Pebbler Game.
- A response to a query $(\mathtt{rekey}, i, j)$ is mapped to "no pebble" if it consists of a valid rekey from $\mathtt{pk}_i$ to $\mathtt{pk}_j$, and to "pebble" otherwise. (Note that this is always well-defined for oblivious PRE reductions, because these need to commit to an assignment of keys at the beginning.)
- Corruption and re-encryption queries are ignored in the Builder-Pebbler Game.
- The challenge query $(\mathtt{challenge}, i^*)$ is mapped to the challenge node T.

Analogously to the adversary in Lemma 34, A is the preimage of B under ϕ: A performs the same rekey queries as B. When B selects a challenge node, A issues a challenge query on the same node with randomly chosen messages $\mathtt{m}_0 \neq \mathtt{m}_1$. A then corrupts all nodes that are not in the challenge graph. If there are any inconsistencies in the corrupted part, A aborts and outputs 0. Finally, A extracts the pebbling configuration $\mathcal{P}$ from the transcript and checks whether the challenge ciphertext is an encryption of $\mathtt{m}_0$ or $\mathtt{m}_1$ under the correct key. If the encrypted message is $\mathtt{m}_0$ (resp. $\mathtt{m}_1$) and the pebbling configuration is a valid node pebbling in the cut defined by $X(G^T)$, then A outputs 0 (resp. 1). Otherwise A outputs always 0. Clearly, this adversary has advantage 1 in the PRE-CPA game.

Since Π is information-theoretically IND-CPA secure, R can only gain any advantage in the IND-CPA game by sending A the challenge ciphertext as response to the challenge query. However, this means the challenge node is associated to the challenge public key. R does not know the corresponding secret key and thus, with overwhelming probability, will respond with a fake rekey when queried for the edge(s) incident on the challenge node. This means in the extracted configuration, the target node is pebbled, so the configuration is not in the cut. Accordingly, the output of A is independent of the IND-CPA challenge bit.

This means, R must attempt to break δ-key privacy. The remaining proof is the same as for Lemma 34, with the δ-key privacy challenge taking the role of the IND-CPA challenge. □

Corollary 16 (Lower bound for PRE restricted to paths)**.** *Let N be the number of users. Then any* oblivious*, key-committing black-box reduction proving adaptive* PRE-CPA *security of a PRE scheme restricted to* paths *based on IND-CPA security and 1-weak key privacy loses at least a factor*

$$\Lambda \geq 2 \cdot N^{\log(N)/8 - \log(\log(N))}.$$

Corollary 17 (Lower bound for PRE restricted to binary trees)**.** *Any* oblivious*, key-committing black-box reduction proving adaptive* PRE-CPA *security of a PRE scheme restricted to* rooted binary in-trees *on N users based on IND-CPA and 2-weak key privacy loses at least a factor*

$$\Lambda \geq N^{\log(N) - \log(\log(N))}.$$

For adversaries that are allowed to query complete (directed acyclic) graphs, Corollary 5 implies an exponential lower bound on the security loss even for non-oblivious black-box reductions:

Corollary 18 (Lower bound for PRE)**.** *Let N be the number of users. Any key-committing, straight-line reduction (possibly non-oblivious) proving unrestricted adaptive* PRE-CPA *security of a PRE scheme based on IND-CPA security and N-weak key privacy loses at least a factor $2^{\Omega(N)}$.*

Handling Rewinding Reductions Theorem 29 does not hold (and thus neither Corollary 5) if we allow Pebbler P to rewind Builder B: P can invoke B once to learn which edges queried in the first phase belong to $\mathcal{P}_v^2$ and $\mathcal{S}_v$, respectively. Then rewind B, and in this 2nd execution the reduction can easily put pebbles so the graph ends up in the cut.

However, Corollary 18 can be extended to rewinding reductions in the following way. We can consider another adversary A* who only at the end of the first phase decides which edges should belong to $\mathcal{P}_v^2$ and $\mathcal{S}_v$. A* will derive the randomness for this assignment by using a random function (only known to A*) on input the transcript of the first query phase. The reduction can get a fresh shot at guessing which edges belong to $\mathcal{P}_v^2$ and $\mathcal{S}_v$ by rewinding A*, but the probability of any such guess being correct is upper bounded as in Equation (7.1) because every time the transcript changes, there's a completely new assignment, and thus the reduction cannot gradually learn anything about the edge assignments.

7.10 Open Problems

We conclude this work by explaining some of the open questions and avenues for further improving our results.

7.10.1 Rewinding Reductions

A large class of reductions that we do not consider in this work are rewinding reductions. While our lower bound for black-box reductions against unrestricted adversaries in the PRE game allows rewinding (see discussion after Corollary 18), this is not the case for our applications of the bounds on edge-pebbling Pebblers. To see this, let us consider the oblivious adversary restricted to paths, which we constructed in the proof of Theorem 25. Since this adversary chooses a uniformly random path in the beginning of the game and then obliviously sticks to this graph structure, we can define a reduction which manages to get into any pebbling configuration it wishes: First, R runs the adversary once on an arbitrary pebbling strategy, e.g., it answers all queries real. Then it rewinds the adversary until the point after it chose the path. But now R knows the full path structure and can trivially embed the pebbles and its challenge such that it ends up in a configuration in the cut.

To fix this issue, we could consider an adversary who follows the same oblivious threshold strategy, but chooses the edges of the path uniformly at random while the game proceeds; i.e., it first chooses a uniform edge $e = (u, v) \leftarrow \mathcal{E} := [1, N]_0 \times [1, N]_0 \setminus \{(x, x) \mid x \in [1, N]_0\}$, then a uniform edge $e' \leftarrow \mathcal{E} \setminus \{(u', v') \mid u' = u \vee v' = v\}$, and so on. In particular, this adversary behaves randomly in each step, conditioned on ending up with a path structure on the set of nodes. However, also this oblivious adversary can be exploited by a rewinding reduction: Assume, R wants to end up with a specific pebbling configuration $\mathcal{P}$. When receiving A's first query, R guesses the position $(i, i + 1)$ of this query on the path. If i is its challenge key it embeds the challenge ciphertext, if $(i, i + 1) \in \mathcal{P}$ it places a pebble, otherwise it answers real. For the next query R rewinds the adversary until it receives a query which is connected to the first edge and, in particular, assuming its initial guess was correct, knows the position of this edge on the path. Thus, it answers this query according to the pebbling configuration it has in mind. R acts similarly for all following queries. If it realises that its initial guess was wrong, R stops and rewinds the adversary until the first query and starts another run of the game. Following this strategy, the reduction has to rewind on expectation $O(N^2)$ times for each of the expected $O(N)$ runs until its initial guess is correct. Thus, this reduction can use the considered adversary at an only polynomial slow-down.

This example shows that assuming non-rewinding (i.e., straight-line) reductions is necessary for our proof to go through, and one can make similar observations for the other adversaries considered in this work. We consider it an interesting open problem to extend (some of) our lower bounds to rewinding reductions. Note that the Builder-Pebbler Game might not be the right abstraction here, since it doesn't capture additional sources of randomness the reduction might choose (e.g. encryption randomness in the case of GSD).

7.10.2 Reductions Exploiting Rewinding or Non-Obliviousness

Instead of extending our lower bounds to rewinding reductions, a promising approach would be to find better reductions using rewinding. It might even be possible to find polynomial reductions for many applications. We believe the GGM prefix-constrained PRF might be a suitable target, due to the general interest of the primitive and because we think this might be technically feasible. Using a similar approach as we laid out for paths in Section 7.10.1 it is easy to see that our lower bound established in Corollary 15 indeed does not hold for rewinding reductions, so this avenue remains open.

Similarly, it would be of interest to find better non-oblivious reductions in the more restricted settings, where we needed to exclude such reductions. This would help clarify in which

settings non-obliviousness is required and where it is simply an artifact of our proof technique. Considering the broader implications, determining the exact restrictions on the adversary that require non-oblivious and/or rewinding reductions could provide guidance towards reductions for new applications that involve dynamic graph-based security games.

7.10.3 Better Reductions for Other Graph Families

We showed in Section 7.4 that the advantage of a Pebbler in the (Restricted) Builder-Pebbler Game is intimately related to cuts in the configuration graph of the challenge graph. Our lower bounds exploited that certain configuration graphs have low weight cuts, such that they are hard to exploit for a reduction. Assume, we could show that for certain graphs there is no such cut in the configuration graph. Could this be exploited to obtain better Pebbler strategies in the Builder-Pebbler Game restricted to such graphs? This has the potential of resulting in better reductions for such graphs in certain applications.

7.10.4 Resoving LKH

The tree-based protocols for CGKA and LKH are extremely related. While we are able to use our techniques to show lower bounds for the former, we are unable to obtain any bounds for LKH as pointed out in Section 7.7. The reason lies in the difference in the security games: while in both games the adversary may force parties into inconsistent states, only in the CGKA game the adversary can exploit this to prompt these parties to generate encryptions to almost arbitrary previous keys. This is not true in the security game for LKH (Multicast Encryption), where all encryptions are generated by a trusted authority, which, by definition, is never in an inconsistent state. Still, the setting is so tantalizingly close to the one of CGKA (and, more generally, GSD on specific graph families), that it seems unlikely that such a bound could not be established. However, at this point we cannot even rule out oblivious reductions, so there might be better reductions lurking even in this class.

CHAPTER 8

Limits on the Adaptive Security of Yao's Garbling

8.1 Introduction

Recall from Chapter 6 that a garbling scheme allows one to garble a circuit C and an input x such that only the output $\mathsf{C}(x)$ can be learned while everything else – besides some leakage such as the size or topology of the circuit – remains hidden. It was originally used by Yao as a means to achieve secure function-evaluation [Yao82, Yao86]. Despite its huge impact on cryptography, it was formally defined as a stand-alone primitive only much later by Bellare, Hoang and Rogaway [BHR12b]. In addition to a syntactic definition, they propose two different security notions for garbling schemes: simulatability and indistinguishability. They show the equivalence of the two definitions[1] in the presence of a *selective* adversary, which sends the circuit and input to be garbled in one shot. In contrast, for the more general case in which the adversary first – in an *offline* phase – chooses a circuit C and then (after receiving its garbling) – in the *online* phase – *adaptively* chooses its input x, the notion of indistinguishability turns out to be strictly weaker than simulatability. Many applications require security in such an adaptive setting, and for the sake of efficiency the cost during the online phase is to be kept minimal.

Prior work on security. Whilst there exist several constructions of provably-secure (even in the adaptive sense) garbling schemes (see related work section 6.1.3 in Chapter 6), a feature of Yao's scheme (and variants thereof) is that security can be proven under the minimal assumption of one-way functions. At the same time, this scheme offers almost-optimal online complexity, with the size of the garbled input being linear in the input-size, and independent of the output- as well as circuit-size. While a formal security proof of Yao's scheme in the *selective* setting was given by Lindell and Pinkas [LP09], Applebaum et al. [AIKW13] showed that the *online complexity* of any adaptively-simulatable garbling scheme must exceed the output-size of the circuit, thereby proving a first limitation of Yao's scheme.

This Chapter essentially replicates, with permission, the full version [KKPW21d] of our publication [KKPW21c], © IACR 2021, https://doi.org/10.1007/978-3-030-84245-1_17.

[1] In the security game for simulatability, the simulator has to simulate $\widetilde{\mathsf{C}}$ given only the output $y = \mathsf{C}(x)$ and some *leakage* $\Phi(\mathsf{C})$. While equivalence of selective simulatability and selective indistinguishability holds for the most natural leakage functions (e.g. the size or topology of C), it *does not* hold for arbitrary leakage functions Φ.

All of this led Jafargholi and Wichs [JW16] to consider a natural adaptation of Yao's garbling scheme (described in Section 8.1.1), where the mapping of output labels to output bits is sent in the online phase as part of the garbled input (see below for the construction). The negative result by Applebaum et al. does not apply to this adaptation of Yao's garbling scheme since its online complexity exceeds the output size. Therefore, this adaptation is the natural version of Yao's garbling scheme for the case of adaptive security, and is the scheme that we consider in this work and will simply refer to as "Yao's garbling" from now on. Jafargholi and Wichs [JW16] were able to show that it satisfies adaptive security for a wide class of circuits, including NC^1 circuits. More precisely, they prove adaptive security of Yao's garbling via a black-box reduction to the IND-CPA security of the underlying symmetric-key encryption (SKE) scheme with a loss in security that is exponential in the *depth* of the circuit. Their proof employs a specially tailored *pebble game* on graphs, and can be seen as an application of the *Piecewise-Guessing framework* (see Section 3.1.3 in Chapter 3 and the more detailed discussion in Chapter 6). Since our work concerns the optimality of this proof, let's recall it in a bit more detail.

8.1.1 Yao's Scheme and Adaptive Indistinguishability

Let's first informally recall Yao's garbling scheme. A circuit $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$ is garbled in the offline phase as follows:

1. For each wire w in C, choose a pair of secret keys $\mathrm{k}_w^0, \mathrm{k}_w^1 \leftarrow \mathsf{Gen}(1^\lambda)$ for a SKE $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

2. For every gate $g : \{0,1\} \times \{0,1\} \to \{0,1\}$ with left input wire u, right input wire v, and output wire w, compute a garbling table $\widetilde{\boldsymbol{g}}$ consisting of the following four ciphertexts (in a random order).

$$\begin{aligned} \mathrm{c}_1 &:= \mathsf{Enc}_{\mathrm{k}_u^0}(\mathsf{Enc}_{\mathrm{k}_v^0}(\mathrm{k}_w^{g(0,0)})) \qquad & \mathrm{c}_2 &:= \mathsf{Enc}_{\mathrm{k}_u^1}(\mathsf{Enc}_{\mathrm{k}_v^0}(\mathrm{k}_w^{g(1,0)})) \\ \mathrm{c}_3 &:= \mathsf{Enc}_{\mathrm{k}_u^0}(\mathsf{Enc}_{\mathrm{k}_v^1}(\mathrm{k}_w^{g(0,1)})) \qquad & \mathrm{c}_4 &:= \mathsf{Enc}_{\mathrm{k}_u^1}(\mathsf{Enc}_{\mathrm{k}_v^1}(\mathrm{k}_w^{g(1,1)})) \end{aligned} \tag{8.1}$$

3. If C has W wires and output wires denoted by $w_{W-m+1}, \ldots, w_W$, assemble the output mapping $\{\mathrm{k}_w^b \to b\}_{i \in [W-m+1, W], b \in \{0,1\}}$.

The garbled circuit $\widetilde{\mathsf{C}}$ consists of all the garbling tables $\widetilde{\boldsymbol{g}}$ as well as the output mapping. To garble an input $x = (b_1, \ldots, b_n)$ in the online phase, simply set

$$\widetilde{x} := (\mathrm{k}_{w_1}^{b_1}, \ldots, \mathrm{k}_{w_n}^{b_n})$$

where w_i denotes the ith input wire. The only difference in the variant from [JW16] is that the sending of the output mapping is moved to the online phase, which leads to an increase in the online complexity to linear in the input- *and* output-size.

To evaluate the garbled circuit on the garbled input, one requires the following *special property* of the SKE: For each ciphertext $\mathrm{c} \leftarrow \mathsf{Enc}_{\mathrm{k}}(\mathrm{m})$ there exists a unique key – namely k – such that decryption doesn't fail. Evaluation of the garbled circuit given the garbled input then works starting from the gates at the lowest level by simply trying which of the four ciphertexts can be decrypted using the two given input keys. This allows to recover exactly one of the two keys associated to the output wire of the respective gate and in the end the output mapping is used to map the sequence of revealed output keys to an output string $y \in \{0,1\}^m$.

Adaptive indistinguishability. A garbling scheme is adaptively indistinguishable if no PPT adversary can succeed in the following experiment[2] with non-negligible advantage:

1. The adversary submits a circuit C to the challenger, who responds with $\widetilde{\mathsf{C}}$.
2. The adversary then submits a pair of inputs (x_0, x_1) such that $\mathsf{C}(x_0) = \mathsf{C}(x_1)$.
3. The challenger flips a coin b and responds with $\widetilde{x}_b$.
4. The adversary wins if it guesses the bit b correctly.

In the following, we will refer to the two games for $b = 0$ and $b = 1$ as the "left" and "right" games, respectively.

To prove adaptive indistinguishability[3] of Yao's scheme for an arbitrary SKE (satisfying the special property), Jafargholi and Wichs construct a black-box reduction from the IND-CPA security of the SKE. More precisely, they proceed by a hybrid argument, where they define a sequence of hybrid games interpolating between the left and the right game such that each pair of subsequent hybrid games only differs in a single ciphertext (in the garbling table) and can be proven indistinguishable by relying on the IND-CPA security of the SKE.

The loss in security incurred by such a reduction then depends on the length of the sequence and the amount of information required to simulate the hybrid games. To end up with a meaningful security guarantee, thus, the sequence of hybrid games must not be too long and it must be possible to simulate any of the hybrid games without relying on too much information, particularly the knowledge of the entire input. Jafargholi and Wichs design such a sequence of hybrid games by using an appropriate pebble game on the topology graph underlying the circuit. In that game, pebble on a gate indicates that the gate is not honestly garbled (as in Equation (8.1)) but is, instead, garbled in some *input-dependent* mode. The pebble rules, which dictate when a pebble can be placed on or removed from a vertex, guarantee that two subsequent hybrids can be proven indistinguishable, and the loss in security directly relates to the number of pebbles on the graph.

Keeping this proof technique in mind, the main idea of this work is to turn a pebble lower bound (w.r.t. an appropriate pebble game) into a lower bound on the security loss inherent to any black-box reduction of adaptive indistinguishability of Yao's scheme. Such an approach we already explored in Chapter 7, also in the context of adaptive security but for primitives that are of a different flavour (e.g., multi-cast encryption). However, the case of garbled circuits turns out very different for several reasons we will highlight later (see Section 8.2.5).

8.1.2 Our Results

We prove a lower bound on the loss in security incurred by *any* black-box reduction from IND-CPA security of the SKE to adaptive indistinguishability of Yao's garbling scheme [JW16]. This immediately implies a similar lower bound with respect to the (stronger) more common

[2] In fact, we define a *weaker* security notion than indistinguishability as defined in [BHR12b] (see Definition 43); according to their definition the adversary can choose *two* circuits $\mathsf{C}_0, \mathsf{C}_1$ of the same topology and inputs x_0, x_1 such that $\mathsf{C}_0(x_0) = \mathsf{C}_1(x_1)$. Aiming at a lower bound on the gap between the security of Yao's scheme and the security of the underlying SKE, the additional restriction we put on our adversary only strengthens our results.

[3] To be precise, [JW16] prove the stronger security notion of simulatability, which implies indistinguishability.

security notion of adaptive simulatability. Our lower bound is subexponential in the depth d of the circuit, hence almost matches the best known upper bound from [JW16].

Theorem (main)**.** *Any black-box reduction from adaptive indistinguishability (and thus also simulatability) of Yao's garbling scheme on the class of circuits with input length n and depth $D \leq 2n$ to the IND-CPA security of the underlying SKE loses at least a factor $\Lambda = \frac{1}{Q} \cdot 2^{\sqrt{D}/61}$, where Q denotes the number of times the reduction rewinds the adversary.*

Two remarks concerning the theorem are in order. Firstly, we are proving a negation of the statement in [JW16], which upper bounds Λ for *every* graph in a class. Therefore, when we say that the class of circuits above loses at least a factor Λ, we mean that there *exists* some circuit Γ in that class such that any reduction loses by that factor (and not that every circuit in that class loses by that factor). The design of this circuit Γ is one of the main technical contributions of this work. The second remark concerns the design of this circuit Γ. In addition to some structural properties that we will come to later, we design Γ to output the constant bit 0. This implies that the output mapping can easily be guessed by a reduction, and therefore the difference, in this case, between Yao's original scheme and [JW16] is only marginal.

Comparison with Applebaum et al. [AIKW13]. The result in [AIKW13] rules out adaptively-simulatable randomised encodings with online complexity less than the output-size of the function it encodes. Since Yao's garbling is one instantiation of randomised encodings, their result immediately rules out its adaptive simulatability. However, [AIKW13] does not apply to our setting for three reasons. Firstly, their result only applies to the original construction of Yao's garbled circuits where the garbled input can be smaller than the output size. In this work we consider the adaptation of Yao's garbling scheme [JW16] where the output mapping is sent in the online phase, hence the online complexity always exceeds the output size. Secondly, their result applies to circuits with large output, while our result holds even for Boolean circuits with outputs of length 1. Finally, their result only applies to simulation security, while our result even holds for indistinguishability.

Comparison with Hemenway et al. [HJO$^+$16]. We would like to emphasise that our lower bound only holds for the *specific* construction of Yao's garbled circuits, and it does not rule out other constructions, even potentially from one-way functions. In fact, the construction of Hemenway et al. already circumvents our result and it is instructive to see how. On a high level, their idea (similar to [BHR12a]) is to take Yao's garbling scheme and then encrypt all the resulting garbling tables with an additional layer of "somewhere equivocal" encryption on top. This change allows them to prove adaptive security with only a *polynomial* loss in security (at the cost of increased online complexity). The intuitive reason why our approach does not apply to this construction is that the additional layer of encryption somehow "blurs out" all the details about the individual garbling tables, on which our argument depends (see *Extracting the Pebble Configuration* in Section 8.2.4).

8.2 Technical Overview

We aim to prove *fine-grained* lower bounds on loss in security incurred by black-box reductions in a setting where a primitive F is used in a protocol Π^F. In our case F is SKE and Π^F is Yao's garbling scheme using the SKE. In order to bound Λ, the loss in security incurred by any

efficient black-box reduction R that breaks F when given black-box access to an adversary that breaks Π^F (i.e., from F to Π^F), we must show that for *every* R, there exists

- an instance F (not necessarily efficiently-implementable) of F and
- an adversary A (not necessarily efficient) that breaks Π^{F}

such that loss in security incurred by R in breaking F is at least Λ.[4] We next describe how the instance and the adversary are defined in our setting.

8.2.1 Our Oracles.

We define two oracles $\mathcal{F}$ and $\mathcal{A}$ implementing an ideal SKE and an adversary, respectively, such that

- the SKE scheme $\mathcal{F} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies IND-CPA security *information-theoretically*,
- the (inefficient) adversary $\mathcal{A}$ breaks indistinguishability of the garbling scheme $\Pi^{\mathcal{F}}$, but is not helpful in breaking the IND-CPA security of $\mathcal{F}$.

Ideal encryption. We will define the ideal SKE oracle $\mathcal{F}$ such that Enc is defined through a random expanding function (which is injective with overwhelming probability). Since the security of $\mathcal{F}$ is information-theoretic, any advantage against IND-CPA which a reduction with oracle access to $\mathcal{F}$ and $\mathcal{A}$ obtains must stem (almost) entirely from the interaction with $\mathcal{A}$. This is true since the reduction only makes polynomially many queries and thus the probability that the answer to one of its oracle queries coincides with the IND-CPA challenge is negligible. On the other hand, a computationally unbounded adversary using an unlimited number of queries can break the scheme and (thanks to injectivity) perfectly recover messages and secret keys from any ciphertext.

The adversary. As for the (inefficient) adversary $\mathcal{A}$, we define a so-called *threshold* adversary which does the following in the indistinguishability game:

1. $\mathcal{A}$ chooses a particular circuit Γ (see Section 8.2.3) which has constant output (bit) 0 and sends Γ to the challenger.
2. After receiving the garbled circuit $\widetilde{\Gamma}$, $\mathcal{A}$ chooses garbling inputs x_0 and x_1 uniformly at random and sends them to the challenger. Note that $\Gamma(x_0) = \Gamma(x_1)$ trivially holds since Γ has constant output.
3. On receipt of the garbled input $\widetilde{x}_b$ along with an output mapping, $\mathcal{A}$ first runs some initial checks on $(\widetilde{\Gamma}, \widetilde{x}_b)$ to verify that the garbling has the correct syntax, and then extracts a *pebble configuration* $\mathcal{P}$ on Γ. That is, every gate in Γ is either assigned a pebble or not, depending on the content of its garbling table in $\widetilde{\Gamma}$ and the garbled input $\widetilde{x}_b$. To compute this mapping, the inefficient adversary $\mathcal{A}$ simply breaks the underlying encryption by brute force. Finally, $\mathcal{A}$ outputs 0 (denoting 'left') if the extracted pebble

[4] This is obtained by simply negating the definition of a black-box reduction: *there exists an efficient reduction* R *for every implementation (not necessarily efficient)* F *of* F *and for every (not necessarily efficient) adversary* A *that breaks* Π^{F} *such that the loss in security is at most* Λ.

configuration is *good* (defined later through some pebble game), and 1 (denoting 'right') otherwise.

By design, the left indistinguishability game (where $b = 0$) will correspond to a good configuration, whereas the right game will not. Therefore the above adversary is a valid distinguisher for the indistinguishability game (Lemma 45). Moreover, $\mathcal{A}$ concentrates all its distinguishing advantage at the *threshold* of good and bad configurations (hence the name). Therefore, loosely speaking, for any reduction to exploit $\mathcal{A}$'s distinguishing advantage, it must somehow embed its own (IND-CPA) challenge at the threshold. All the technicality in proving our main theorem goes into formalising this intuition, which we summarise next in Section 8.2.2.

8.2.2 High-Level Idea

To prove a lower bound on Λ (Theorem 32), we construct a *punctured* adversary $\mathcal{A}[\mathtt{c}^*]$ (see Section 8.3.5) which behaves similar to $\mathcal{A}$ *except* when it comes to the hardcoded challenge ciphertext $\mathtt{c}^* \leftarrow \mathsf{Enc}_{\mathtt{k}^*}(\mathtt{m})$ (for some arbitrary message $\mathtt{m}$). We aim to puncture $\mathcal{A}[\mathtt{c}^*]$ such that it never decrypts $\mathtt{c}^*$ but instead just proceeds by assuming that $\mathtt{c}^*$ decrypts to the all-0 string, and hence cannot be of any help to a reduction that aims to break $\mathtt{c}^*$. However, we have to be careful here since the reduction embedding $\mathtt{c}^*$ in $\widetilde{\Gamma}$ will also embed other ciphertexts under key $\mathtt{k}^*$ (which it can derive through querying its IND-CPA encryption oracle $\mathsf{Enc}_{\mathtt{k}^*}$), and hence $\mathcal{A}[\mathtt{c}^*]$ would learn the key $\mathtt{k}^*$ when brute-force decrypting these ciphertexts. We solve this issue by endowing $\mathcal{A}[\mathtt{c}^*]$ with a decryption oracle $\mathsf{Dec}_{\mathtt{k}^*}$ that allows to find and decrypt those ciphertexts under $\mathtt{k}^*$. Since our ideal encryption scheme actually satisfies the stronger notion of IND-CCA security, this decryption oracle is of no help to the reduction.

The core of our lower bound is now to define the circuit Γ and the notion of *good* pebble configurations such that the following holds:

- Our threshold adversary $\mathcal{A}$ indeed breaks the garbling scheme.
- It is hard to distinguish $\mathcal{A}$ from $\mathcal{A}[\mathtt{c}^*]$.

For the latter property, note that any efficient reduction R can only distinguish $\mathcal{A}$ from $\mathcal{A}[\mathtt{c}^*]$ if their outputs differ, which only happens if they extract different pebbling configurations $\mathcal{P} \neq \mathcal{P}^*$ such that one of them is good and the other bad. Thus, to bound the success probability of R, it suffices to establish the following two properties:

1. The pebbling configurations $\mathcal{P}$ and $\mathcal{P}^*$ extracted by $\mathcal{A}$ and $\mathcal{A}[\mathtt{c}^*]$ (in the same execution of the game, using the same randomness) differ by at most one valid pebbling move in some natural pebble game[5], where a pebble can be placed on or removed from a gate if at least one of its parent gates carries a pebble.
2. It is hard for any reduction to produce $(\widetilde{\Gamma}, \widetilde{x})$ such that $\mathcal{A}$ extracts a *threshold* configuration, i.e. a pebble configuration that is good but can be switched to a bad configuration within one valid pebbling move.

[5] In Section 8.3.3 we actually consider a much more finegrained pebble game, where different types of pebbles represent different garbling modes of a gate. For this exposition, it suffices to focus on this simplified game.

Intuitively, pebbles on gates in the circuit represent malformed gates, i.e., gates whose garbling table is different from the honest garbling table. When considering circuits consisting only of non-constant gates, the pebbling rule in Property 1 captures the fact that a reduction cannot produce ciphertexts encrypting the key $\mathtt{k}^*$ under which its challenge ciphertext $\mathtt{c}^* \leftarrow \mathsf{Enc}_{\mathtt{k}^*}(\mathtt{m})$ (for some arbitrary $\mathtt{m}$) was encrypted. Hence, in order to embed $\mathtt{c}^*$ at a gate, the reduction has to first output a malformed garbling (not encoding $\mathtt{k}^*$) for its predecessor gate. Now, to see why Property 1 holds – i.e., the pebbling configurations $\mathcal{P}$ and $\mathcal{P}^*$ extracted by $\mathcal{A}$ and $\mathcal{A}[\mathtt{c}^*]$ follow the same dynamics – note that the behaviour of $\mathcal{A}$ and $\mathcal{A}[\mathtt{c}^*]$ can only differ if $\mathtt{k}^*$ is not encrypted in any ciphertext.

The tricky part of our proof is to establish Property 2 which, on a high level, works as follows. For a reduction R to simulate a threshold configuration we first force it to maul – and hence pebble – several gates. Then, for this mauling to go 'undetected' we force R to correctly guess the value of these gates when Γ is evaluated at x_0. This, intuitively, will be the source of its loss. To this end, we design our circuit Γ to consist of two blocks[6], $\Gamma^{\oplus}$ and $\Gamma^{\wedge}$. Looking ahead, whether there is a pebble on a gate in $\Gamma^{\oplus}$ will be *independent* of the input and correspond to R's attempt at guessing x_0 (this relies on the properties of XOR gates). The pebbles on $\Gamma^{\wedge}$, in contrast, will be extractable with respect to the input garbling $\widetilde{x}_b$ and indicate whether or not the guesses on x_0 in the $\Gamma^{\oplus}$ block were correct (this relies on the properties of AND gates). Moreover, by definition:

- In case of a proper garbling of (Γ, x_0) (i.e., the left game), the adversary $\mathcal{A}$ will not extract any pebble on $\Gamma^{\oplus}$ or $\Gamma^{\wedge}$.
- In case of a proper garbling of (Γ, x_1) (i.e., the right game), on the other hand, the adversary $\mathcal{A}$ will not extract any pebbles on $\Gamma^{\oplus}$, but will extract some pebbles on $\Gamma^{\wedge}$ (since $x_1 \neq x_0$).

Accordingly, we define the *good* predicate such that the empty configuration is good, whereas any configuration containing a pebble on $\Gamma^{\wedge}$ is bad, and therefore the above ensures that $\mathcal{A}$ breaks the security of the garbling scheme. Furthermore, the threshold configurations contain many pebbles on $\Gamma^{\oplus}$, but no pebbles on $\Gamma^{\wedge}$. In other words, threshold configurations require R to make many guesses about x_0 and all of them need to be correct, which is unlikely to occur. This establishes Property 2.

8.2.3 The Circuit Γ and the Good Predicate

The design of topology of the circuit $\Gamma^{\oplus}$ is such that it has high pebbling complexity with respect to our pebble game: i.e., every valid pebbling sequence starting from the *initial* empty configuration and reaching a *final* configuration that has a pebble on an output gate of $\Gamma^{\oplus}$, must contain a "heavy" configuration with many, say d, pebbles. To guarantee that threshold configurations contain many pebbles, we define the good configurations as those that are reachable with $d-1$ pebbles following valid pebbling moves. Since $\Gamma^{\wedge}$ will (topologically) succeed $\Gamma^{\oplus}$ in Γ, any configuration with a pebble on $\Gamma^{\wedge}$ is in particular bad (since an output gate of $\Gamma^{\oplus}$ must have been pebbled first). At the same time, to allow for our "control mechanism", we construct Γ so that each gate g in $\Gamma^{\oplus}$ has a 'companion' successor gate in $\Gamma^{\wedge}$ that helps check correctness of g's output. Thus for each AND gate in $\Gamma^{\wedge}$, one of the

[6]For this high-level overview, we ignore the third block Γ^0 consisting of a binary tree of AND gates, whose sole purpose is to guarantee constant 0 (bit) output.

inputs comes from the output of $\Gamma^{\oplus}$ and the other from the output of its companion gate (see Figure 8.1). This fixes the topology of Γ and we choose the type of gate as to enforce Property 2, as explained below.

- The $\Gamma^{\oplus}$ circuit is composed only of XOR gates, since these gates allow us to maintain *high entropy* (of the input), and hence guarantee that it is hard to guess the outputs of the pebbled gates in $\Gamma^{\oplus}$ (see Section 8.3.2). Furthermore, XOR gates are *symmetric* with respect to their input in the sense that from the garbling table alone even an inefficient adversary cannot distinguish which keys are associated with which bits. This property allows $\mathcal{A}$ to extract the pebbling configuration of $\Gamma^{\oplus}$ just from $\widetilde{\Gamma}$, independently of the input (see next section).
- The $\Gamma^{\wedge}$ circuit, on the other hand, is composed of AND gates. Since AND gates are *asymmetric* (since only $(1,1)$ maps to 1, while all three other input pairs map to 0), we can use them to detect errors in the $\Gamma^{\oplus}$ circuit: i.e., looking at a garbling table of an AND gate our adversary $\mathcal{A}$ can exploit this asymmetry to easily associate keys to bits. Thus, whenever during evaluation of $\widetilde{\Gamma}$ on input $\widetilde{x}$ the adversary $\mathcal{A}$ receives wrong input keys for a (properly garbled) AND gate, $\mathcal{A}$ considers this gate as malformed and associates it with a pebble. (The case of AND gates which are not properly garbled is rather technical and we refer the reader to Section 8.3.4.)

8.2.4 Extracting the Pebble Configuration

Since it is central to the working of our adversary $\mathcal{A}$ (and is a somewhat subtle matter), here we provide a high-level description of the extraction mechanism.[7] First of all, recall that pebbles on $\Gamma^{\oplus}$ and $\Gamma^{\wedge}$ have different meanings: a pebbled XOR gate indicates that its garbling table is *malformed* whereas a pebbled AND gate indicates that R's guess for the companion XOR gate is *wrong*. This, coupled with the fact that the gates have differently-structured gate tables (i.e., symmetric vs. asymmetric) means that the extraction mechanism for the two gates (and hence the blocks) is also different. In particular, as we will see, the pebble status of an XOR gate is something that can inferred solely from the garbled circuit $\widetilde{\Gamma}$ (and thus can be done in the offline phase) whereas the pebble status of an AND gate is something that also depends on the garbled input $\widetilde{x}$ and is necessarily done in the online phase. Let's look at how the respective extraction is carried out. First, given $\widetilde{\Gamma}$, $\mathcal{A}$ extracts a key pair for each wire in Γ from the encryptions associated with its *successor* gates, or the output mapping; if this cannot be done uniquely, $\mathcal{A}$ aborts and outputs 1 (we refer to Section 8.3.4 for more details). In the following, for a gate g, let u and v denote the input wires, w the output wire, and k_u, k'_u, k_v, k'_v, k_w, k'_w the corresponding keys associated with these wires.

- If g is an XOR gate, then the honest garbling table of g can be derived from Equation (8.1) as

$$\begin{array}{ll} \mathsf{Enc}_{\mathrm{k}_u}(\mathsf{Enc}_{\mathrm{k}_v}(\mathrm{k}_w)) & \mathsf{Enc}_{\mathrm{k}'_u}(\mathsf{Enc}_{\mathrm{k}_v}(\mathrm{k}'_w)) \\ \mathsf{Enc}_{\mathrm{k}_u}(\mathsf{Enc}_{\mathrm{k}'_v}(\mathrm{k}'_w)) & \mathsf{Enc}_{\mathrm{k}'_u}(\mathsf{Enc}_{\mathrm{k}'_v}(\mathrm{k}_w)). \end{array}$$

[7] In Section 8.3.4 we consider a more general extraction mechanism that can be extended to arbitrary gates and assigns different types of pebbles, representing the "distance" of a garbling table $\widetilde{g}'$ for a gate g from an honest garbling table $\widetilde{g}$. For ease of exposition, here we consider a simplified pebble game and only discuss how to extract pebbles for XOR and AND gates, where a pebble in this simplified game would correspond to different sets of pebbles for XOR and AND gates in the more fine-grained pebble game.

Whenever a garbling table $\widetilde{g}$ differs from this representation (i.e., not symmetric), $\mathcal{A}$ assigns g a pebble and this assignment is *independent* of the bits running over the wires u, v, w and the keys revealed during evaluation. Thus, $\mathcal{A}$ can extract pebbles on $\Gamma^{\oplus}$ already *before* it chose the inputs x_0, x_1, in particular independently of $\widetilde{x}$.

- For an AND gate g, on the other hand, the garbling table of g consists of four ciphertexts derived from Equation (8.1) as

$$\begin{aligned} &\mathsf{Enc}_{\mathtt{k}_u}(\mathsf{Enc}_{\mathtt{k}_v}(\mathtt{k}_w)) \qquad &&\mathsf{Enc}_{\mathtt{k}'_u}(\mathsf{Enc}_{\mathtt{k}_v}(\mathtt{k}_w)) \\ &\mathsf{Enc}_{\mathtt{k}_u}(\mathsf{Enc}_{\mathtt{k}'_v}(\mathtt{k}_w)) \qquad &&\mathsf{Enc}_{\mathtt{k}'_u}(\mathsf{Enc}_{\mathtt{k}'_v}(\mathtt{k}'_w)). \end{aligned}$$

  Since the roles of the keys are *asymmetric*, the pebble extraction will depend on the bits b_u, b_v, b_w running over the wires and the keys $\mathtt{k}^r_u$, $\mathtt{k}^r_v$, $\mathtt{k}^r_w$ revealed during evaluation. A first attempt would be to simply map keys to bits as $\mathtt{k}_u, \mathtt{k}_v, \mathtt{k}_w \to 0$ and $\mathtt{k}'_u, \mathtt{k}'_v, \mathtt{k}'_w \to 1$, and assign g a pebble if $\mathtt{k}^r_\eta \not\to b_\eta$ for some $\eta \in \{u, v, w\}$. Unfortunately, this simple idea does not work since a reduction R might embed its challenge ciphertext $\mathtt{c}^* \leftarrow \mathsf{Enc}_{\mathtt{k}^*}(\mathtt{m})$ in the garbling of an AND gate (recall from Section 8.2.3 that the gates in $\Gamma^{\wedge}$ receive one input from an output gate of $\Gamma^{\oplus}$ and the other input from their companion gate *within* the circuit $\Gamma^{\oplus}$). Now, if R embeds the challenge key $\mathtt{k}^*$ at an output wire of $\Gamma^{\oplus}$, it must pebble an output gate in $\Gamma^{\oplus}$, hence end up with a bad pebbling configuration independently of $\mathtt{c}^*$. However, this is not true if R embeds $\mathtt{k}^*$ at the other input wire of the AND gate. Thus, $\mathcal{A}$ must not extract a pebble for a garbling table that can be derived from an honest garbling table by embedding a challenge key at this wire. We show in Section 8.3.4 that such malformed garblings of AND gates either involve guessing the input bits or they can still be used for our "control mechanism".

8.2.5 Comparison with Chapter 7

While here as well as in Chapter 7 we model choices made by a reduction by putting pebbles on a graph structure, the analogy basically ends there. In Chapter 7 an interactive game between a "builder" and a "pebbler" is considered in which the builder chooses edges and the pebbler decides adaptively whether to pebble them. The goal of the pebbler is to get into a "good" configuration, and the difficulty for the reduction (playing the role of the pebbler) there lies in the fact that the graph is only revealed edge-by-edge. In contrast, in the setting of garbling the graph structure is initially known and the game has just two rounds. The difficulty for the reduction here comes from having to guess the bits running over a subset of wires during evaluation of the circuit. None of the main ideas from Chapter 7 seem applicable in this setting and vice versa. For example, many of the results in Chapter 7 are restricted to the limited class of so-called oblivious reductions, while our setting doesn't share the difficulties encountered in Chapter 7; in particular, our result holds for arbitrary (even rewinding) black-box reductions.

8.3 Lower Bound for Yao's Garbling Scheme

For the notation and definitions, we refer to Section 6.2 in Chapter 6.

Let Π denote the variant of Yao's garbling scheme as analyzed in [JW16]. In this section, we aim to prove a lower bound on the loss in security involved when proving adaptive indistinguishability of Π based on the IND-CPA security of the underlying symmetric encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. As explained in the introduction, we follow the approach in Chapter 7

and define two oracles $\mathcal{F}$ and $\mathcal{A}$ implementing an ideal SKE scheme and an adversary, respectively, such that $\mathcal{A}$ is not helpful in breaking IND-CPA security of $\mathcal{F}$. For the precise description of $\mathcal{F}$ we refer to Section 8.3.5. The (inefficient) threshold adversary $\mathcal{A}$ we define as follows:

1. On input the security parameter in unary, 1^λ, the adversary $\mathcal{A}$ chooses a circuit Γ with input size $n = \Theta(\lambda)$, constant output, and depth $D(d) \in O(n)$ for a parameter d. The circuit Γ consists of three parts, i.e., $\Gamma = \Gamma^0 \circ \Gamma^\wedge \circ \Gamma^\oplus$. $\mathcal{A}$ sends Γ to the challenger.
2. After receiving $\widetilde{\Gamma}$, the adversary $\mathcal{A}$ chooses $x_0, x_1 \leftarrow \{0,1\}^n$ uniformly at random. Note that $\Gamma(x_0) = \Gamma(x_1)$ trivially holds since Γ has constant output. $\mathcal{A}$ sends x_0, x_1 to the challenger.
3. On receipt of $\widetilde{x}_b = (\mathrm{k}_1, \ldots, \mathrm{k}_n)$ along with an output mapping, $\mathcal{A}$ extracts a *pebbling configuration* on the graph $\Gamma \setminus \Gamma^0$ corresponding to $\Gamma^\wedge \circ \Gamma^\oplus$ as described in Section 8.3.4. $\mathcal{A}$ outputs $b' = 0$ if the pebbling configuration is *good* as per Definition 74, and $b' = 1$ otherwise.

We will first provide a precise definition of the candidate circuit Γ in Section 8.3.1 and then show the following two properties of this circuit: First, in Section 8.3.2 we will prove that if a large subset of gates in $\Gamma^\oplus$ is malformed, then on uniformly random input some of these gates will not evaluate correctly. Second, in Section 8.3.3, we introduce a new pebbling game on DAGs and prove a pebbling lower bound on the graph Γ underlying Γ. The definition of *good* pebbling configurations in Definition 74 then gives a cut in the configuration graph of $\Gamma \setminus \Gamma^0$ w.r.t. this new pebbling game. Having proven these properties of the circuit, in Section 8.3.4 we will then describe a mapping from garbled circuit/input pair $(\widetilde{\Gamma}, \widetilde{x})$ to a pebbling configuration on $\Gamma \setminus \Gamma^0$. This mapping together with the cut in the configuration graph will guarantee that the threshold adversary $\mathcal{A}$ indeed breaks indistinguishability of the garbling scheme $\Pi^\mathcal{F}$. Finally, in Section 8.3.5, it then remains to combine these results. First, we will essentially prove that any black-box reduction proving adaptive indistinguishability of the garbling scheme based on the IND-CPA security of the underlying encryption scheme must *follow the pebbling rules*, i.e., it must define two hybrid games such that the extracted pebbling configurations differ by one valid pebbling move; this step will crucially rely on our choice of gates. The pebbling lower bound from Section 8.3.3 then implies that any threshold configuration contains many pebbles on $\Gamma^\oplus$. We will then use the result from Section 8.3.2 as well as the technical fix from Section 8.3.4 concerning *equivocation* of keys to show that the simulation of any garbling $(\widetilde{\Gamma}, \widetilde{x})$ that is mapped to a threshold configuration requires to guess many input bits. This will allow us to state our final theorem.

8.3.1 The Circuit

We construct a family of circuits $\{\Gamma_d\}_{d\in\mathbb{N}}$ and show that the loss in security for Γ_d is exponential in $\sqrt{d}$, a parameter linear in the depth D. The circuit is designed keeping our high-level idea in mind, we denote its underlying graph by Γ_d. The circuit $\Gamma_d := \Gamma_d^0 \circ \Gamma_d^\wedge \circ \Gamma_d^\oplus$ consists of the three blocks $\Gamma_d^\oplus$, $\Gamma_d^\wedge$ and Γ_d^0, with underlying graphs denoted by $\Gamma_d^\oplus$, $\Gamma_d^\wedge$ and Γ_d^0, respectively. The graph $\Gamma_d^\oplus$ (see Figure 8.2.(b)) is a so-called *tower graph* [DKW11a], and is obtained from so-called pyramid graphs of depth d (see Figure 8.2.(a)).

- $\Gamma_d^\oplus$ is obtained from $\Gamma_d^\oplus$ by substituting each vertex with an XOR gate as shown in Figure 8.2. On a high level, the pyramid structure ensures high pebbling complexity

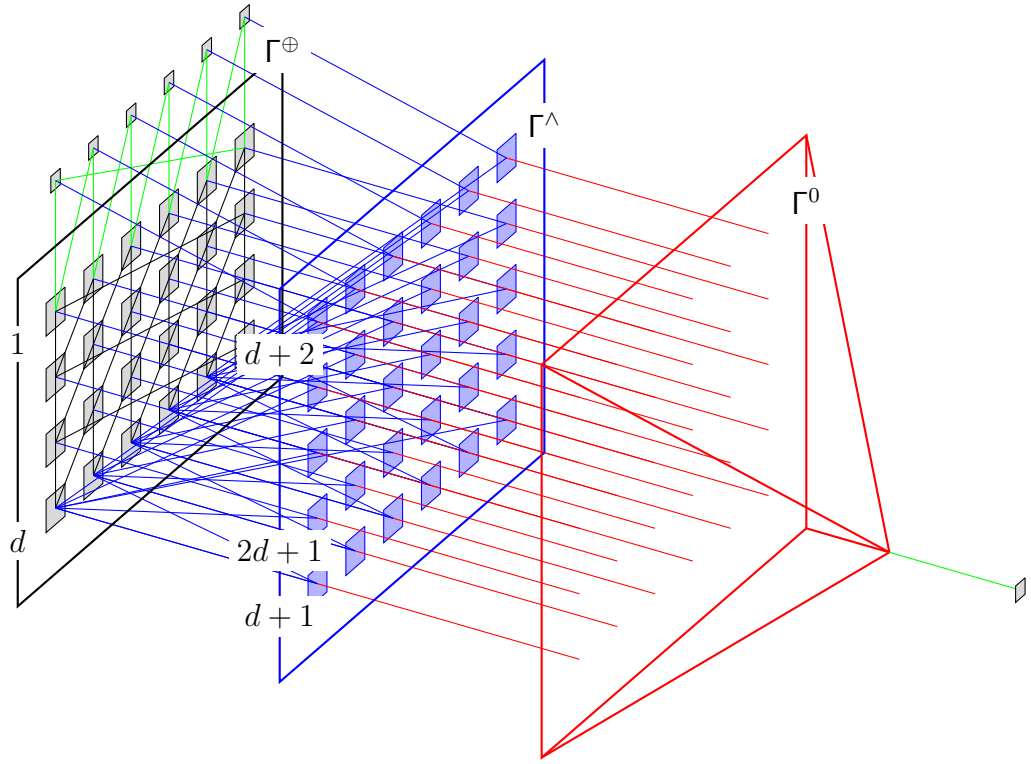


Figure 8.1: A schematic diagram for the candidate circuit of width 5 and depth 4. The input and output wires are coloured green. The layer number is indicated on the left. The first two blocks are the XOR and AND layers respectively; the final pyramid denotes the binary tree.

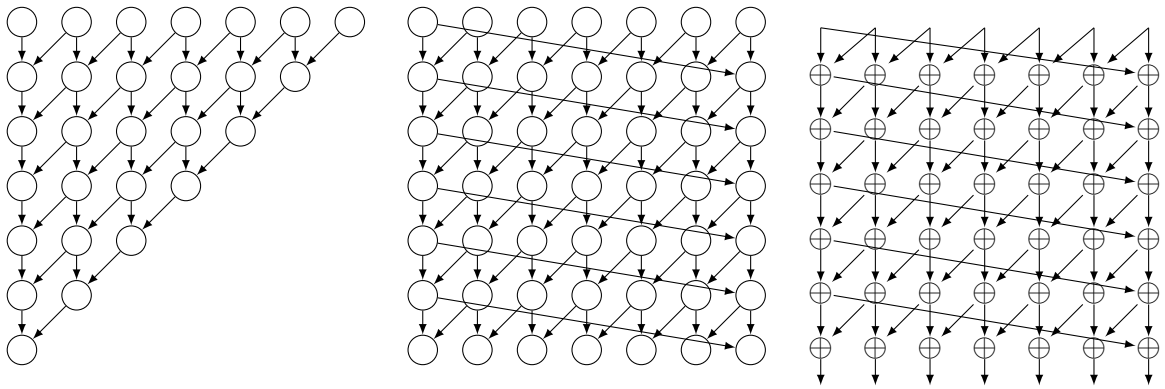
whereas the XOR gates preserve (most) entropy in the input, which makes it hard for a reduction to obtain correct evaluation of pebbled gates.

- Γ^0_d consists of a binary tree of AND gates and its sole role is to set the output of the circuit Γ to constant 0 (Lemma 40).[8]

- $\Gamma^{\wedge}_d$ sits in between the $\Gamma^{\oplus}_d$ and Γ^0_d blocks (see Figure 8.1), and consists of one AND gate serving as "control" gate for each XOR gate in $\Gamma^{\oplus}_d$ and each input gate. Each AND gate g in $\Gamma^{\wedge}_d$ receives its inputs from (i) the output of its companion XOR gate in $\Gamma^{\oplus}_d$ (resp. input gate) and (ii) the XOR gate in the last layer of $\Gamma^{\oplus}_d$ in (vertical) alignment with g (see Figure 8.1, formal definition below). As mentioned previously, intuitively, this block will act as an "error detection" mechanism for the $\Gamma^{\oplus}_d$ block in the sense that it helps detect if (malformed) garblings of XOR gates evaluate wrongly.

More formally, for input size $n = 2^{\kappa} - 1$ with $\kappa \in \mathbb{N}$, and $d \leq n$, we describe the candidate circuit $\Gamma = \Gamma_d$ based on its underlying graph structure $G = G_d$ as follows, see Figure 8.1: Γ contains $D(d) := 2d + \lceil \log((d+1)n) \rceil + 2$ layers, each containing n nodes. The input gates (layer 0) have outdegree 2, the nodes on layers $[1, 2d+1]$ all have in- and outdegree 2, and the nodes in the last $\lceil \log(d \cdot n) \rceil$ layers have indegree 2 and outdegree 1. Since we will need to differentiate between left and right parents a node, we will define the edge sets of these graphs as the union of "right" and "left" edges. The first $d+1$ layers $[0, d]$ build a graph $\Gamma^{\oplus}$

[8] In principle we could have used constant-0 gates in place of the AND gates, or simply a single constant-0 gate of high fan-in (which would considerably simplify the description). But we prefer to stick to the standard Boolean basis.



(a) (b) (c)

Figure 8.2: The graphs and the circuit for parameter $d = 6$: (a) A pyramid graph of depth d, (b) Extending the pyramid graph to get a tower graph $\Gamma_d^{\oplus}$ of depth d and (c) Circuit $\Gamma_d^{\oplus}$ obtained replacing the vertices in $\Gamma_d^{\oplus}$ with XOR gates.

of high pebbling complexity (see Section 8.3.3), defined as

$$\begin{aligned}
\Gamma^{\oplus} :=& ([0, (d+1)\cdot n - 1], \mathcal{E}^{\oplus}), \text{ where } \mathcal{E}^{\oplus} := \mathcal{E}_L^{\oplus} \cup \mathcal{E}_R^{\oplus} \text{ with} \\
\mathcal{E}_L^{\oplus} :=& \{((i-1)\cdot n + k, i\cdot n + k) \mid i \in [1,d]\,,\ k \in [0,n]\}, \\
\mathcal{E}_R^{\oplus} :=& \{((i-1)\cdot n + l, i\cdot n + k) \mid i \in [1,d]\,,\ k,l \in [0,n]\,,\ l = k+1 \mod n\},
\end{aligned}$$

where we number the input gates by $\{0, \ldots, n-1\}$. Graph $\Gamma^{\oplus}$ is followed by $d+1$ layers $[d+1, 2d+1]$ building $\Gamma^{\wedge}$, defined as

$$\begin{aligned}
\Gamma^{\wedge} :=& ([(d+1)\cdot n, (2d+2)\cdot n - 1], \mathcal{E}^{\wedge}), \text{ where } \mathcal{E}^{\wedge} := \mathcal{E}_L^{\wedge} \cup \mathcal{E}_{\mathsf{in},R}^{\wedge} \cup \mathcal{E}_R^{\wedge} \text{ with} \\
\mathcal{E}_L^{\wedge} :=& \{(d\cdot n + k, (d+i)\cdot n + k) \mid i \in [1,d+1]\,,\ k \in [0,n]\}, \\
\mathcal{E}_{\mathsf{in},R}^{\wedge} :=& \{(d\cdot n + l, (d+1)\cdot n + k) \mid k,l \in [0,n]\,,\ l = k+1 \mod n\}, \\
\mathcal{E}_R^{\wedge} :=& \{((i-2)\cdot n + k, (d+i)\cdot n + k) \mid i \in [2,d+1],\ k \in [0,n]\}.
\end{aligned}$$

Finally, there is a binary tree structure Γ^0 on top of the $(d+1)\cdot n$ output gates of $\Gamma^{\wedge}$, to guarantee constant output 0 of the circuit.

The candidate circuit Γ is now defined based on the graph structure Γ as follows:

- All gates on layers $[1, d]$ implement XOR gates.
- All other gates consist of AND gates.

In the following lemma we prove that Γ is indeed constant.

Lemma 40. $\Gamma(x) = 0$ *for all* $x \in \{0,1\}^n$*, i.e.,* Γ *is constant.*

Proof. To see that this circuit has constant output 0, first note that Γ^0 outputs 1 only on the all-1 string 1^n. In Section 8.3.2 we will provide an explicit representation of the output of $\Gamma^{\oplus}$ that in particular implies that the range of $\Gamma^{\oplus}$ consists of strings $(y_1, \ldots, y_n) \in \{0,1\}^n$ that contain an even number of 1s (see Corollary 19). As we chose $n = 2^{\kappa} - 1$ odd, this implies that any output of $\Gamma^{\oplus}$ must contain at least one 0. Hence, the input to $\Gamma^{\wedge}$ contains at least one 0, and since $\Gamma^{\wedge}$ only contains AND gates, the output of $\Gamma^{\wedge}$ must contain a 0. This proves that Γ is constant. □

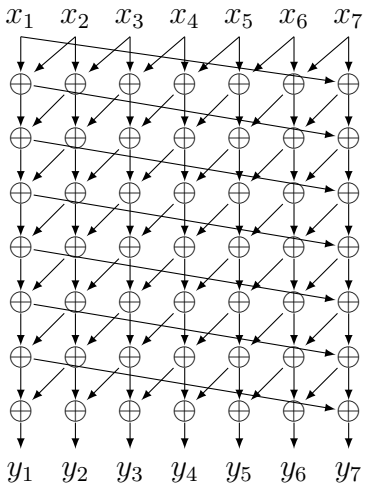


Figure 8.3: Circuit $\Gamma^{\oplus}$ with $n = d = 7$.

8.3.2 Vulnerability of the Circuit $\Gamma^{\oplus}$

In Section 8.3.5 we will prove that any black-box reduction R that aims to use $\mathcal{A}$ to gain advantage in breaking the IND-CPA security of encryption scheme $\mathcal{F}$ has to simulate $(\widetilde{\Gamma}, \widetilde{x})$ such that the extracted pebbling configuration on $\Gamma^{\oplus}$ contains $d-1$ or d grey or black pebbles. Each of these pebbles implies that at least one of the ciphertexts associated to that gate must be malformed and modify the output of some input key pair. In the case that all AND gates are properly garbled, all keys can be mapped to bits and hence such a switch of the output can be detected (cf. Lemma 46). Thus, we consider the following game.

- On input a circuit C and a parameter d, R chooses a circuit C′ of the same topology as C such that all except exactly d (non-input) gates coincide with the corresponding gates in C. R sends C′ to $\mathcal{A}$.
- On receipt of C′, $\mathcal{A}$ samples $x \leftarrow \{0,1\}^n$ uniformly at random.
- R wins if for all gates in C′ the output during evaluation on input x coincides with the corresponding output bit when evaluating C.

We now prove that for $\mathsf{C} = \Gamma^{\oplus}$, no algorithm R wins the above game with non-negligible (in d) probability.

Lemma 41. *Let $d \in [1, n]$. For $\mathsf{C} = \Gamma^{\oplus}$ and any* R*, the probability that* R *wins the above game is at most $(\frac{3}{4})^{\sqrt{d}/4}$.*

First, note that all except d gates in C′ are XOR gates, and in particular a linear function over $\mathbb{Z}_2$. For each of the remaining d malformed gates, on the other hand, at least one input pair is mapped to a different output bit than it would be in an XOR operation. We call the corresponding gates in the original circuit $\Gamma^{\oplus}$ *pebbled*. To prove Lemma 41, we will show that there exists a subset of at least $\sqrt{d}/4$ of those d pebbled gates such that their input is determined by independent linear functions. This implies that instead of choosing $x \leftarrow \{0,1\}^*$, $\mathcal{A}$ can equivalently choose the $\sqrt{d}/2$ input bits uniformly at random, and then choose x uniformly under the constraint that the values running over these wires during evaluation of

$\Gamma^{\oplus}$ must be consistent with the predetermined bits. Clearly, x chosen this way is still uniformly random in $\{0,1\}^n$. By definition of the game, R only wins the game if for all gates in C′ the output during evaluation on input x coincides with the corresponding output bit when evaluating C, and this must in particular also hold for the pebbled gates. Since each of the malformed gates in C′ flips the output of at least one of the four possible input pairs, and the input bits of $\sqrt{d}/4$ of the pebbled gates were chosen independently and uniformly at random, the probability that R wins is at most $(\frac{3}{4})^{\sqrt{d}/4}$.

Towards proving Lemma 41, let M denote the linear mapping corresponding to one layer of gates in the circuit $\Gamma^{\oplus}$, i.e., written in matrix notation,

$$M = \begin{pmatrix} 1 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 1 & \dots & 0 & 0 & 0 \\ \vdots & & & \ddots & & & \vdots \\ 0 & 0 & 0 & \dots & 0 & 1 & 1 \\ 1 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix}.$$

The output of the μth layer of $\Gamma^{\oplus}$ on input $x \in \{0,1\}^n$ is given by $M^{\mu} \cdot x$, hence we denote the degree-1 polynomial in $\mathbb{Z}_2[x_1, \dots, x_n]$ which determines its ν-th bit by M^{μ}_{ν} (for $\mu \in [0,n]$ and $\nu \in [1,n]$). Denoting by $\overline{\nu+1}$ the representation of the residue class $\nu + 1 \mod n$ in $[1,n]$, we have e.g.,

$$M^0_{\nu} = x_{\nu}, \quad M^1_{\nu} = x_{\nu} \oplus x_{\overline{\nu+1}}, \quad M^2_{\nu} = x_{\nu} \oplus x_{\overline{\nu+2}}, \quad M^3_{\nu} = x_{\nu} \oplus x_{\overline{\nu+1}} \oplus x_{\overline{\nu+2}} \oplus x_{\overline{\nu+3}}$$

and in general it holds

$$M^{\mu}_{\nu} = M^{\mu-1}_{\nu} \oplus M^{\mu-1}_{\overline{\nu+1}} \tag{8.2}$$

for all $\mu, \nu \in [1,n]$. In the following we will associate gates with the corresponding polynomials that determine their outputs.

If the input length n is odd – for convenience we assume n to be one less than a power of 2 – then $\Gamma^{\oplus}$ maintains high entropy; to prove this, we use the following explicit representation of the polynomials M^{μ}_{ν}.

Lemma 42 (explicit formula for the polynomials M^{μ}_{ν})**.** *Let $n = 2^{\kappa} - 1$, $\kappa \in \mathbb{N}$, M defined above, $\mu \in \mathbb{N}$, and $\nu \in [1,n]$. For $\overline{\mu} \neq n$ and $\beta_k \in \{0,1\}$ its binary decomposition, i.e. $\overline{\mu} = \sum_{k \in [0,\kappa-1]} \beta_k 2^k$, it holds:*

$$M^{\mu}_{\nu} = \bigoplus_{i \in [1,n]} \alpha_i x_i, \text{ where } \alpha_i = \begin{cases} 1 & \text{if } i \in \nu + \sum_{k \in [0,\kappa-1]} \{0, \beta_k\} \cdot 2^k \mod n, \\ 0 & \text{else.} \end{cases} \tag{8.3}$$

Note, M^{μ}_{ν} only depends on $\overline{\mu}$, not on μ. For $\overline{\mu} = n = 2^{\kappa} - 1$, it holds:

$$M^{\mu}_{\nu} = \bigoplus_{i \in [1,n]} \alpha_i x_i, \text{ where } \alpha_i = \begin{cases} 1 & \text{if } i \neq \nu, \\ 0 & \text{else.} \end{cases} \tag{8.4}$$

Proof. We prove the claim via induction on $\mu \in \mathbb{N}$. For $\mu = 1$, we have $M^{\mu}_{\nu} = x_{\nu} \oplus x_{\overline{\nu+1}}$. On the other hand, for $\mu = 1$ we have $\beta_0 = 1$ and $\beta_k = 0$ for all $k \in [1,\kappa]$, which implies $\alpha_{\nu} = 1$, $\alpha_{\overline{\nu+1}}$ and $\alpha_i = 0$ for all $i \in [1,n] \setminus \{\nu, \overline{\nu+1}\}$. Hence, the claim is true for $\mu = 1$.

For $2 \leq \mu \leq n-1$, let $\mu - 1 = \sum_{k\in[0,\kappa-1]} \beta'_k 2^k$, hence for $\mu = \sum_{k\in[0,\kappa-1]} \beta_k 2^k = \sum_{k\in[0,\kappa-1]} \beta'_k 2^k + 1$ we obtain

$$\beta_k = \begin{cases} 1-\beta'_k & \text{for } k \leq k' := \min\{k \in [0,\kappa-1] \mid \beta'_k = 0\}, \\ \beta'_k & \text{for } k > k'. \end{cases}$$

By induction hypothesis, we have

$$M_\nu^{\mu-1} = \bigoplus_{i\in[1,n]} \alpha_i^{(0)} x_i, \text{ where } \alpha_i^{(0)} = \begin{cases} 1 & \text{if } i \in \mathcal{I}^{(0)} := \{\nu + \sum_{k\in[0,\kappa-1]} [0,\beta'_k]\cdot 2^k \mod n\}, \\ 0 & \text{else.} \end{cases}$$

$$M_{\nu+1}^{\mu-1} = \bigoplus_{i\in[1,n]} \alpha_i^{(1)} x_i, \text{ where } \alpha_i^{(1)} = \begin{cases} 1 & \text{if } i \in \mathcal{I}^{(1)} := \{\nu + 1 + \sum_{k\in[0,\kappa-1]} [0,\beta'_k]\cdot 2^k \mod n\}, \\ 0 & \text{else.} \end{cases}$$

Let $\mathcal{I}^{(0)} \Delta \mathcal{I}^{(1)} := (\mathcal{I}^{(0)} \setminus \mathcal{I}^{(1)}) \cup (\mathcal{I}^{(1)} \setminus \mathcal{I}^{(0)})$ denote the symmetric difference of $\mathcal{I}^{(0)}$ and $\mathcal{I}^{(1)}$. Then, by Equation (8.2), we get

$$M_\nu^\mu = \bigoplus_{i\in[1,n]} \alpha_i x_i \quad \text{with } \alpha_i = \alpha_i^{(0)} \oplus \alpha_i^{(1)} = \begin{cases} 1 & \text{if } i \in \mathcal{I}^{(0)} D \mathcal{I}^{(1)}, \\ 0 & \text{else.} \end{cases}$$

Since $\beta'_{k'} = 0$ and $\beta'_k = 1$ for $k < k'$, we have for $0 \leq k < k'$:

$$\nu + 1 + \sum_{l=0}^{k-1} 2^l + \sum_{l=k+1}^{\kappa-1} [0,\beta'_l]\, 2^l \mod n = \nu + 2^k + \sum_{l=k+1}^{\kappa-1} [0,\beta'_l]\, 2^l \mod n \in \mathcal{I}^{(1)} \cap \mathcal{I}^{(0)},$$

and for $k = k'$:

$$\nu + 1 + \sum_{l=0}^{k'-1} 2^l + \sum_{l=k'+1}^{\kappa-1} [0,\beta'_l]\, 2^l \mod n = \nu + 2^{k'} + \sum_{l=k'+1}^{\kappa-1} [0,\beta'_l]\, 2^l \mod n \in \mathcal{I}^{(1)} \setminus \mathcal{I}^{(0)},$$

$$\text{and} \quad \nu + \sum_{l=k'+1}^{\kappa-1} [0,\beta'_l]\, 2^l \mod n \in \mathcal{I}^{(0)} \setminus \mathcal{I}^{(1)}.$$

Combining the above cases and using that $\beta_{k'} = 1$ and $\beta_k = 0$ for $k < k'$, proves Equation (8.3) for $\mu \in [1, n-1]$.

To prove Equation (8.4), note that for $\mu - 1 = n - 1 = 2^\kappa - 2 = \sum_{k\in[1,\kappa-1]} 2^k$, Equation (8.3) gives

$$M_\nu^{\mu-1} = \bigoplus_{i\in[1,n]} \alpha_i^{(0)} x_i, \text{ where } \alpha_i^{(0)} = \begin{cases} 1 & \text{if } i \in \mathcal{I}^{(0)} := \{\nu + 2\cdot[0,(n-1)/2] \mod n\} \\ & \qquad = \{\nu, \nu+2, \ldots, \nu-1\}, \\ 0 & \text{else.} \end{cases}$$

$$M_{\nu+1}^{\mu-1} = \bigoplus_{i\in[1,n]} \alpha_i^{(1)} x_i, \text{ where } \alpha_i^{(1)} = \begin{cases} 1 & \text{if } i \in \mathcal{I}^{(1)} := \{\nu + 1 + 2\cdot[0,(n-1)/2] \mod n\} \\ & \qquad = \{\nu+1, \nu+3, \ldots, \nu\}, \\ 0 & \text{else.} \end{cases}$$

Using $\mathcal{I}^{(0)} \Delta \mathcal{I}^{(1)} = \{\nu + [1, n-1] \mod n\} = [1,n] \setminus \{\nu\}$ now proves Equation (8.4).

Finally, for $\mu = 2^\kappa = n + 1$, Equation (8.4) implies

$$M_\nu^{2^\kappa} = M_\nu^n \oplus M_{\overline{\nu+1}}^n = \left(\bigoplus_{i\in[1,n]\setminus\{\nu\}} x_i\right) \oplus \left(\bigoplus_{i\in[1,n]\setminus\{\overline{\nu+1}\}} x_i\right) = x_\nu \oplus x_{\overline{\nu+1}} = M_\nu^1,$$

where we used the fact that $n = 2^\kappa - 1$ by definition. This proves the Lemma. □

Lemma 42 directly implies several useful properties, which we summarize in the following corollary.

Corollary 19 (Properties of M and $\Gamma^{\oplus}$)**.** *For M defined as above, $n = 2^\kappa - 1$, $\kappa \in \mathbb{N}$, it holds*

- $M^{2^\kappa} = M$*, which implies* $\mathsf{rank}(M^k) = n - 1$ *for all $k \geq 1$, i.e., $\Gamma^{\oplus}$ is 2-to-1 for any d.*
- *Any $n-1$ output bits of M^k ($k \geq 1$) are determined by linearly independent degree-1 polynomials.*
- $\mathsf{Image}(\Gamma^{\oplus}) = \{x = (x_1, \ldots, x_n) \in \{0,1\}^n \mid \bigoplus_{i\in[1,n]} x_i = 0\}$*, i.e., all vectors in the image of $\Gamma^{\oplus}$ contain an even number of 1s.*

The first property immediately follows from Lemma 42 since for $\mu = 2^\kappa$ we have $\overline{\mu} = 1$. The second property then follows from $\mathsf{rank}(M^k) = n - 1$. For the last property, note that the set $\nu + \sum_{k\in[0,\kappa-1]}\{0, \beta_k\} \cdot 2^k \mod n$ is even whenever a single bit β_k is nonzero (which is true for all $\overline{\mu} > 0$), and also the set $\{i \in [1,n] \mid i \neq \nu\}$ is even since n is odd.

The following Lemma now immediately implies Lemma 41.

Lemma 43. *Any subset $\mathcal{S} \subset \{M_\nu^\mu\}_{\mu\in[0,n],\nu\in[1,n]}$ of polynomials in $\mathbb{Z}_2[x_1, \ldots, x_n]$ with $s := |\mathcal{S}|$ contains a subset $\mathcal{S}'$ of size $\sqrt{s}/4$ such that $|\mathsf{parents}(\mathcal{S}')| = \sqrt{s}/2$ and $\mathsf{parents}(\mathcal{S}')$ is linearly independent, where $\mathsf{parents}(M_\nu^\mu) := \{M_\nu^{\mu-1}, M_{\nu+1}^{\mu-1}\}$.*

Proof. We split the $(n+1) \times n$ gates $\{M_\nu^\mu\}_{\mu\in[0,n],\nu\in[1,n]}$ into four equal-sized quarters $\mathcal{M}_i, i \in [1,4]$, each containing a subset of $(n+1)/2 \times (n+1)/2$ gates, see Figure 8.4. Since $\mathcal{S}$ has size s, at least one of these quarters must contain $\geq s/4$ gates from $\mathcal{S}$. Furthermore, considering the $(n+1)/2$ vertical paths within such a quarter; then either 1) there is one vertical path which contains $\geq \sqrt{s}/2$ gates from $\mathcal{S}$, or 2) there exist $\geq \sqrt{s}/2 + 1$ vertical paths which contain at least one pebble each.

For case 1), note that for all $\nu \in [1,n]$ the set of gates within any vertical path within $\mathcal{M}_i$, i.e., $\{M_\nu^\mu\}_{\mu\in[0,n]} \cap \mathcal{M}_i = \{M_\nu^{\mu_i+j}\}_{j\in[0,(n-1)/2]}$ with $\mu_1 = \mu_2 = 0,\ \mu_3 = \mu_4 = (n+1)/2$, is linearly independent. To see this, first note that by Corollary 19 $\{M_\nu^{\mu_i}, \ldots, M_{\nu+(n-1)/2}^{\mu_i}\}$ is linearly independent for any $i \in [1,4]$ and generates the vertical path $\{M_\nu^{\mu_i+j}\}_{j\in[0,(n-1)/2]}$ in $\mathcal{M}_i$. Now, consider the explicit representation of $M_\nu^{\mu_i+j}$ with $j = \sum_{k\in[0,\kappa-2]} \beta_k 2^k$ in the basis $\{M_\nu^{\mu_i}, \ldots, M_{\nu+(n-1)/2}^{\mu_i}\}$ (which follows from Equation (8.3)):

$$M_\nu^{\mu_i+j} = \bigoplus_{l\in[0,(n-1)/2]} \alpha_l M_{\nu+l}^{\mu_i}, \text{ where } \alpha_l = \begin{cases} 1 & \text{if } \ l \in \nu + \sum_{k\in[0,\kappa-2]} [0,\beta_k] \cdot 2^k \mod n, \\ 0 & \text{else.} \end{cases}$$

In particular, it follows that $M_\nu^{\mu_i+j} = \bigoplus_{l\in[0,j]} \alpha_l M_{\nu+l}^{\mu_i}$ with $\alpha_j = 1$ and $\alpha_l = 0$ for all $l \in [j+1, (n-1)/2]$. This implies that for $b_j \in \{0,1\}^*$

$$\bigoplus_{j\in[0,(n-1)/2]} b_j \cdot M_\nu^{\mu_i+j} = 0 \quad \Rightarrow \quad b_j = 0 \quad \forall j \in [0, (n-1)/2]\,.$$

Thus, any subset of gates $\{M_\nu^{\mu_i+j}\}_{j\in[0,(n-1)/2]}$ along the ν-th vertical path in $\mathcal{M}_i$ is linearly independent over $\mathbb{Z}_2$. This implies that the set of $\geq \sqrt{s}/2$ gates in $\mathcal{S}$ which lie on one vertical path is linearly independant. It immediately follows that the left parents of this set are linearly independent as well. By basic linear algebra, replacing an element from a set of linearly

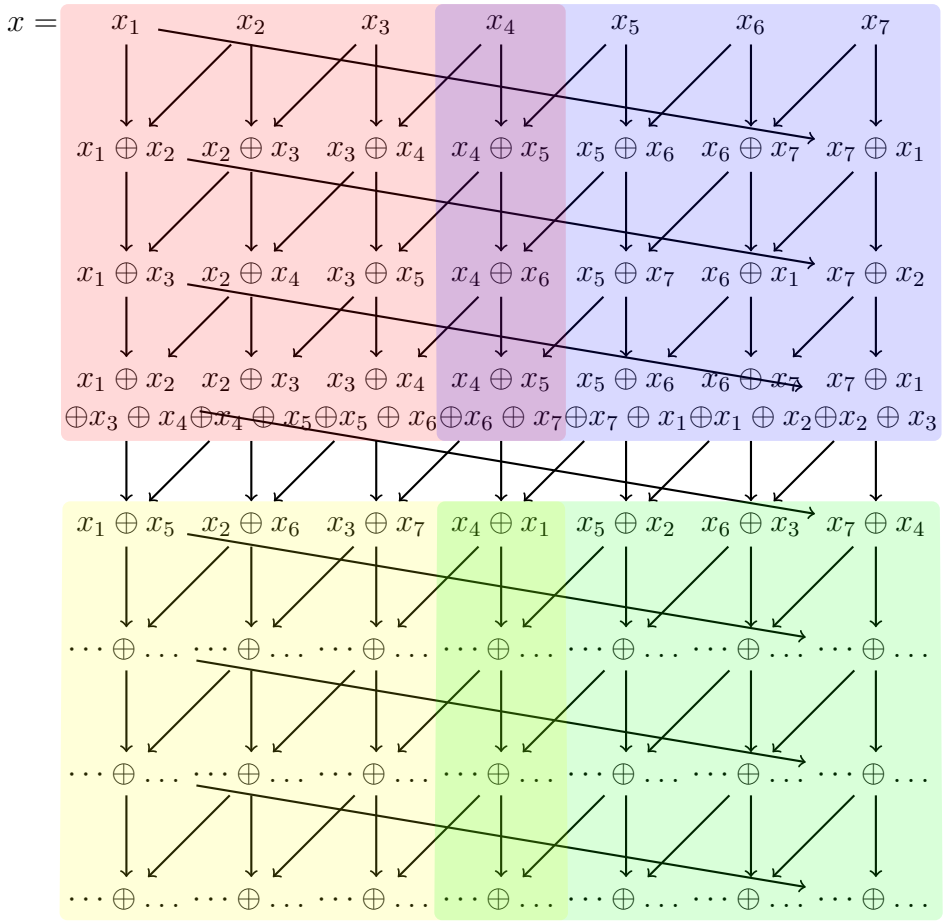


Figure 8.4: The circuit $\Gamma^{\oplus}$ split into four equal-sized quarters.

independent equations by a linear combination of this element with other elements of the set preserves linear independence. Using Equation (8.2) and removing at most half of the left parents, we obtain a set $\mathcal{S}'$ of $\geq \sqrt{s}/4$ gates whose parents are distinct and form a linearly independent set.

For case 2), by assumption there exists a set $\mathcal{S}'$ consisting of $\sqrt{s}/4$ gates in $\mathcal{S}$ such that *their parents* lie on distinct vertical paths in $\mathcal{M}_i$.[9] Furthermore, since $\geq \sqrt{s}/2 + 1$ vertical paths contain gates from $\mathcal{S}$, we can choose $\mathcal{S}'$ such that parents$(\mathcal{S}')$ does not contain the bottom right gate $M^{\mu_i+(n-1)/2}_{\nu_i+(n-1)/2}$ with $\nu_1 = \nu_3 = 1$, $\nu_2 = \nu_4 = (n-1)/2$ (which is not necessary but more convenient for the analysis below). We will now argue that the set of parents of $\mathcal{S}'$ is linearly independent. Similar to above, we can uniquely represent the elements of parents$(\mathcal{S}') := \{M^{\mu_i+\mu_j}_{\nu_i+\nu_j}\}_{j\in[1,\sqrt{s}/2]}$ with $\mu_j, \nu_j \in [0, (n-1)/2]$, and $\nu_j < \nu_{j+1}$ for all $j \in [1, \sqrt{s}/2 - 1]$ as a linear combination of the linearly independent set $\{M^{\mu_i}_{\nu_i}, \ldots, M^{\mu_i}_{\nu_i+n-1}\}$:

$$M^{\mu_i+\mu_j}_{\nu_i+\nu_j} = \bigoplus_{l\in[0,\mu_j]} \alpha_l M^{\mu_i}_{\nu_i+\nu_j+l} \quad \text{with } \alpha_0 = 1 \text{ and } \mu_j + \nu_j \leq n-1.$$

This implies that for $b_j \in \{0,1\}^*$

$$\bigoplus_{j\in[1,\sqrt{s}/2]} b_j \cdot M^{\mu_i+\mu_j}_{\nu_i+\nu_j} = 0 \quad \Rightarrow \quad b_j = 0 \quad \forall j \in [1, \sqrt{s}/2].$$

Hence, $\mathcal{S}'$ is indeed linearly independent over $\mathbb{Z}_2$. This proves the claim. □

[9] Technically, for $i \in \{3,4\}$ we have to shift the window $\mathcal{M}_i$ by setting $\mu_i \leftarrow \mu_i - 1$.

Lemma 41 now follows, since for any set of d pebbled gates, by Lemma 43 there exists a subset $\mathcal{S}'$ of $\sqrt{d}/4$ pebbled gates such that their parents are distinct and form a linearly independent set.

8.3.3 Pebbling Game and Threshold

Recall that in Yao's garbling scheme, each gate g is associated with a (honest) garbling table $\widetilde{\boldsymbol{g}}$, which consists of four double encryptions that encode g's gate table. However, a reduction is free to alter the contents of the honest garbling table in any way. In fact, the upper bounds in [LP09, JW16] crucially rely on the ability to do this in an indistinguishable manner: in the real game the garbling tables are all honest, whereas in the simulated game the garbling tables all encode the constant-0 gate, and the hybrids involve replacing the honest garbling tables one by one with that of the constant-0 gate.[10] We introduce a pebble game to precisely model such different simulations of the garbled circuit $\widetilde{\Gamma}$ (by the reduction). Loosely speaking, the extracted pebble configuration is an abstract representation of the simulation $(\widetilde{\Gamma}, \widetilde{x}_b)$, and the pebbling rules model the reduction's ability to maul garbling tables in $\widetilde{\Gamma}$ without being noticed (indistinguishability).

The pebbles. Intuitively, the pebble on a gate g encodes how "different" the garbling table $\widetilde{\boldsymbol{g}}'$ which $\mathcal{A}$ receives is from an honest garbling $\widetilde{\boldsymbol{g}}$. To this end, we employ three different pebbles: white, grey and black.

- A white pebble on g indicates that $\widetilde{\boldsymbol{g}}'$ and $\widetilde{\boldsymbol{g}}$ are at "distance" 0 (defined below), i.e., $\widetilde{\boldsymbol{g}}$ is (distributed identically to) an honest garbling table of g.
- A grey or black pebble on g indicates that $\widetilde{\boldsymbol{g}}'$ is malformed. What differentiates grey from black is the degree of malformation: loosely speaking, a grey pebble indicates that $\widetilde{\boldsymbol{g}}'$ is at a distance 1 from $\widetilde{\boldsymbol{g}}$, whereas a black pebble indicates that $\widetilde{\boldsymbol{g}}'$ is at a distance 2 (or more).

To understand what we mean by distance, we need to take a closer look at the structure of a garbling table. An honest garbling table $\widetilde{\boldsymbol{g}}$ consists of the four double encryptions shown in Table 8.1.(a). We assign a grey pebble to a gate g if the garbling table of g in $\widetilde{\Gamma}$ can be proven indistinguishable from $\widetilde{\boldsymbol{g}}$ by embedding a *single* IND-CPA challenge key (among k_u^0, k_u^1, k_v^0 and k_v^1). For example, let's consider an AND gate and its honest garbling table (Table 8.1.(b)): a malformed table that is at distance one (via the key k_u^1 or k_v^1) from it is, e.g., a garbling table that encodes the constant-0 gate (Table 8.1.(d)). A garbling of an XOR gate, in contrast, is at distance 2 from a garbling of a constant gate: If k_u^a and k_v^b are the keys revealed during evaluation, then the garbling of an XOR gate can be proven indistinguishable from the constant-$(a \oplus b)$ gate only by first embedding a challenge key at k_u^{1-a} and then a second challenge key at k_v^{1-b}, or vice versa; i.e. the reduction needs to embed challenges at each input wire.

Pebbling rules. To complete the description of a pebble game, we need to describe the pebbling rules. These rules essentially capture the following observation: a reduction (with overwhelming probability) cannot possess encryptions of its (IND-CPA) challenge key.

[10] Note, this simulation crucially relies on the fact that keys can be *equivocated*: While the output keys are all associated to 0, when altering the output mapping accordingly evaluation will still succeed. Note that in the selective setting for Yao's original scheme as well as in the adaptive setting for the modified scheme [JW16] the input is known before the output mapping is sent.

$\mathsf{E}_{\mathsf{k}_u^0}(\mathsf{E}_{\mathsf{k}_v^0}(\mathsf{k}_w^{g(0,0)}))$ $\mathsf{E}_{\mathsf{k}_u^1}(\mathsf{E}_{\mathsf{k}_v^0}(\mathsf{k}_w^{g(1,0)}))$ $\mathsf{E}_{\mathsf{k}_u^0}(\mathsf{E}_{\mathsf{k}_v^1}(\mathsf{k}_w^{g(0,1)}))$ $\mathsf{E}_{\mathsf{k}_u^1}(\mathsf{E}_{\mathsf{k}_v^1}(\mathsf{k}_w^{g(1,1)}))$	$\mathsf{E}_{\mathsf{k}_u^0}(\mathsf{E}_{\mathsf{k}_v^0}(\mathsf{k}_w^{0}))$ $\mathsf{E}_{\mathsf{k}_u^1}(\mathsf{E}_{\mathsf{k}_v^0}(\mathsf{k}_w^{0}))$ $\mathsf{E}_{\mathsf{k}_u^0}(\mathsf{E}_{\mathsf{k}_v^1}(\mathsf{k}_w^{0}))$ $\mathsf{E}_{\mathsf{k}_u^1}(\mathsf{E}_{\mathsf{k}_v^1}(\mathsf{k}_w^{1}))$	$\mathsf{E}_{\mathsf{k}_u^0}(\mathsf{E}_{\mathsf{k}_v^0}(\mathsf{k}_w^{0}))$ $\mathsf{E}_{\mathsf{k}_u^1}(\mathsf{E}_{\mathsf{k}_v^0}(\mathsf{k}_w^{1}))$ $\mathsf{E}_{\mathsf{k}_u^0}(\mathsf{E}_{\mathsf{k}_v^1}(\mathsf{k}_w^{1}))$ $\mathsf{E}_{\mathsf{k}_u^1}(\mathsf{E}_{\mathsf{k}_v^1}(\mathsf{k}_w^{0}))$	$\mathsf{E}_{\mathsf{k}_u^0}(\mathsf{E}_{\mathsf{k}_v^0}(\mathsf{k}_w^{0}))$ $\mathsf{E}_{\mathsf{k}_u^1}(\mathsf{E}_{\mathsf{k}_v^0}(\mathsf{k}_w^{0}))$ $\mathsf{E}_{\mathsf{k}_u^0}(\mathsf{E}_{\mathsf{k}_v^1}(\mathsf{k}_w^{0}))$ $\mathsf{E}_{\mathsf{k}_u^1}(\mathsf{E}_{\mathsf{k}_v^1}(\mathsf{k}_w^{0}))$
(a)	(b)	(c)	(d)

Table 8.1: Garbling tables for (a) general gate g, (b) AND gate, (c) XOR gate, and (d) constant-0 gate. u and v denote the two input wires, whereas w denotes the output wire.

Therefore, whenever the garbling table $\widetilde{\boldsymbol{g}}$ of a gate g has been switched to a malformed garbling $\widetilde{\boldsymbol{g}}'$ (say) at distance one, (at least) one of the garbling tables associated to its predecessor gates, say g_u, *must* have been first switched to a garbling that encodes only one of g_u's output keys. This is required to "free up" one of g_u's output keys (so that it can now be set as the challenge key). Looking ahead, we will be interested in pebbling the circuit $\Gamma^{\oplus}$ which consists of XOR gates only. Hence, the pebbling rules are designed to capture the structure of XOR gates. Recall that an XOR gate is at distance 2 from a constant gate, thus, we end up with the following rules (where g_u and g_v denote the two predecessors of g):

1. a grey pebble can be placed on or removed from a gate g only if (at least) one of its predecessor gates (say g_u) carries a black pebble; and
2. a grey pebble on a gate g can be swapped with a black pebble if the *other* predecessor gate (i.e., g_v) carries a black pebble.

The actual game. The above white-grey-black ($\mathtt{WGB}$) pebble game is a simplified version of the ($\mathtt{WG^3B}$) pebble game we end up using, but it is sufficient to convey the essential ideas that we use. The actual game, defined in Definition 73 (Section 8.3.3), is more fine-grained: in order to keep track of the inner and outer encryptions, we introduce three types of grey pebbles (grey-left, grey-right and grey-free), and the pebbling rules are also modified accordingly.

Definition 73 (Reversible $\mathtt{WG^3B}$ pebbling game for indegree-2 graphs)**.** Consider a directed acyclic graph $\Gamma = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = [1, N]$ and let $\mathcal{X} = \{\mathtt{W}, \mathtt{G}_*, \mathtt{G}_L, \mathtt{G}_R, \mathtt{B}\}$ denote the set of colours of the pebbles. Consider a sequence $\boldsymbol{\mathcal{P}} := (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ of pebbling configurations for Γ, where $\mathcal{P}_i \in \mathcal{X}^{\mathcal{V}}$ for all $i \in [0, \tau]$. We call such a sequence *a $\mathtt{WG^3B}$ pebbling strategy* for Γ if the following two criteria are satisfied:

1. In the initial configuration all the vertices are pebbled white (i.e., $\mathcal{P}_0 = (\mathtt{W}, \ldots, \mathtt{W})$) and in the final configuration *at least one sink* of $\mathtt{G}$ is pebbled grey (i.e., $\mathcal{P}_\tau = (\ldots, \mathtt{G}_\cdot, \ldots)$), where $\mathtt{G}_\cdot$ denotes an arbitrary type of grey, i.e. $\mathtt{G}_\cdot \in \{\mathtt{G}_*, \mathtt{G}_L, \mathtt{G}_R\}$.

2. Two subsequent configurations differ only in one vertex and the following rules are respected in each move:

   a) $\mathtt{W} \leftrightarrow \mathtt{G}_*$: a white pebble can be replaced by a $\mathtt{G}_*$ pebble (and vice versa) if one of its parents is black-pebbled

   b) $\mathtt{W}/\mathtt{G}_* \leftrightarrow \mathtt{G}_L$: a white or $\mathtt{G}_*$ pebble can be replaced by a $\mathtt{G}_L$ pebble (and vice versa) if its *left* parent is black-pebbled

   c) $\mathtt{W}/\mathtt{G}_* \leftrightarrow \mathtt{G}_R$: a white or $\mathtt{G}_*$ pebble can be replaced by a $\mathtt{G}_R$ pebble (and vice versa) if its *right* parent is black-pebbled

d) $\mathtt{G}_L \leftrightarrow \mathtt{B}$: a $\mathtt{G}_L$ pebble can be replaced by a black pebble (and vice versa) if its *right* parent is black-pebbled

e) $\mathtt{G}_R \leftrightarrow \mathtt{B}$: a $\mathtt{G}_R$ pebble can be replaced by a black pebble (and vice versa) if its *left* parent is black-pebbled

The *space-complexity* of a $\mathtt{WG}^3\mathtt{B}$ pebbling strategy $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\tau)$ for a DAG Γ is defined as

$$\sigma_\Gamma(\boldsymbol{\mathcal{P}}) := \max_{i\in[0,\tau]} |\{j \in [1,N] : \mathcal{P}_i(j) \in \{\mathtt{G}_*, \mathtt{G}_L, \mathtt{G}_R, \mathtt{B}\}\}|.$$

For a subgraph Γ' induced on vertex set $\mathcal{V}' \subset \mathcal{V}$, the space-complexity of $\boldsymbol{\mathcal{P}}$ restricted to Γ' is defined as

$$\sigma_{|\Gamma'}(\boldsymbol{\mathcal{P}}) := \max_{i\in[0,\tau]} |\{j \in \mathcal{V}' : \mathcal{P}_i(j) \in \{\mathtt{G}_*, \mathtt{G}_L, \mathtt{G}_R, \mathtt{B}\}\}|.$$

The space-complexity of a DAG Γ is the minimum space-complexity over all of its strategies $\boldsymbol{\mathcal{P}}^\Gamma$:

$$\sigma(\Gamma) := \min_{\boldsymbol{\mathcal{P}}\in\boldsymbol{\mathcal{P}}^\Gamma} \sigma_\Gamma(\boldsymbol{\mathcal{P}}). \tag{8.5}$$

Remark 15. Note that for upper bounds the $\mathtt{G}_*$ pebbles would be redundant in the following sense: any $\mathtt{WG}^3\mathtt{B}$ pebbling sequence including $\mathtt{G}_*$ pebbles can be replaced by a valid $\mathtt{WG}^3\mathtt{B}$ sequence (potentially including redundant steps) that does not contain $\mathtt{G}_*$ pebbles and has a smaller or equal space-complexity. The reader familiar with pebbling games might notice that – ignoring the $\mathtt{G}_*$ pebbles – our $\mathtt{WG}^3\mathtt{B}$ pebbling rules exactly correspond to *reversible edge-pebbling* (see Definition 8 in Chapter 3) : In this game, pebbles are placed on edges instead of nodes, and a pebble can be placed on/removed from an edge $(u, v) \in \mathcal{E}$ if and only if all edges incident on u are pebbled.

The following lemma gives a lower bound on the $\mathtt{WG}^3\mathtt{B}$ pebbling complexity of the graph $\Gamma \setminus \Gamma^0$ underlying the first two blocks $\mathsf{\Gamma}^\wedge \circ \mathsf{\Gamma}^\oplus$ of our candidate circuit $\mathsf{\Gamma}$.

Lemma 44 (Pebbling lower bound on $\Gamma \setminus \Gamma^0$)**.** *Let $\Gamma \setminus \Gamma^0$ be the graph underlying the circuit $\mathsf{\Gamma}^\wedge \circ \mathsf{\Gamma}^\oplus$. To grey-pebble a gate on layer $d' \in [1, d+1]$ following the reversible $\mathtt{WG}^3\mathtt{B}$ pebbling rules from Definition 73, one requires space-complexity at least $d'-1$. Furthermore, to $\mathtt{G}_L$- or $\mathtt{B}$-pebble a gate on layer $d' \geq d+1$, one requires at least d grey or black pebbles simultaneously on the first d layers.*

Proof. We rely on the crucial observation that the ancestor graph of any vertex v in layer $d' \in [1, d+1]$ of Γ forms a so-called pyramid graph of depth d', with v as the unique sink. Let's denote this graph by $D_{d'}$. To prove the lemma, first note, that pebbling any node on layer d' requires to *black*-pebble one of its parents (see Figure 8.5). Thus, to prove the Lemma, it suffices to argue that it takes space-complexity $\geq d'-1$ to place a black pebble on the sink of $D_{d'-1}$. We will prove the following slightly stronger claim:

Claim 3. *Any $\mathtt{WG}^3\mathtt{B}$ pebbling sequence $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_L)$ on $D_{d'}$ with unique sink v^*, where $\mathcal{P}_0 = \{\mathtt{W}, \ldots, \mathtt{W}\}$ is the all-white configuration and $\mathcal{P}_L = \{\ldots, \mathtt{B}\}$ is a configuration where the sink of $D_{d'}$ is black-pebbled, contains a pebbling configuration $\mathcal{P}^*$ such that $\sigma(\mathcal{P}^*) \geq d'$ and each path from a source to v^* contains at least one grey or black pebble.*

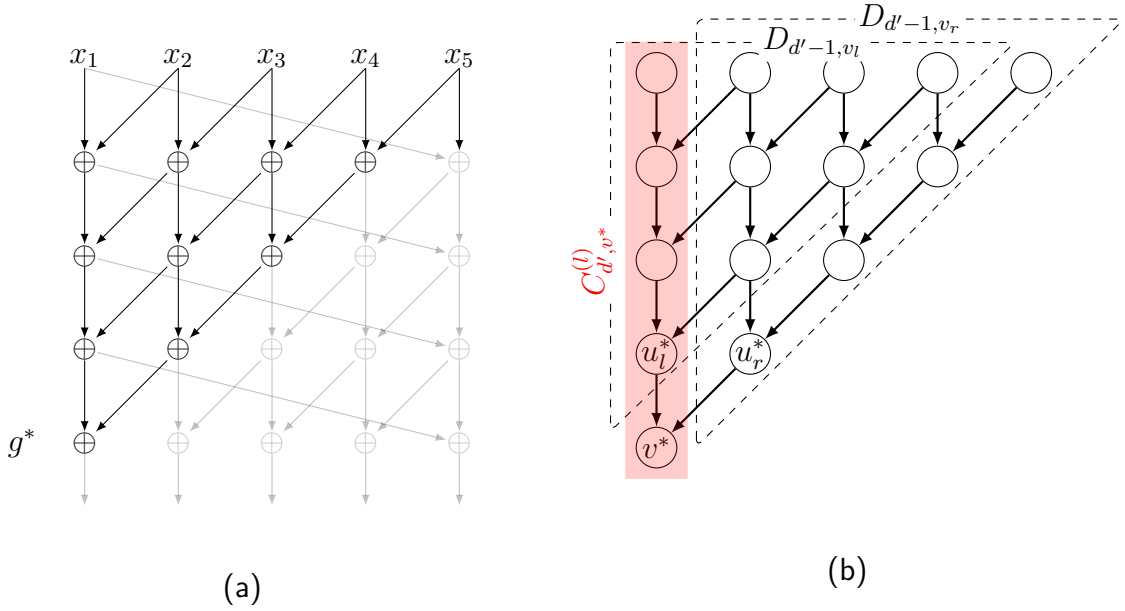


Figure 8.5: Pyramid graph used in the proof of Lemma 44. (a) The gates in $\Gamma^{\oplus}$ from which the gate g^* can be reached are highlighted (b) The correponding pyramid graph $D_{d'}$. The subgraphs of interest are highlighted.

We prove the claim via induction on d'. For $d' = 1$ the claim is obviously true. For the induction step, note that, to black-pebble node v^*, one needs to black-pebble its parents v_l and v_r before, not necessarily at the same time though. Let $D_{d'-1,v_l}$, $D_{d'-1,v_r}$ denote the ancestor graphs of v_l and v_r, respectively, each being pyramid graphs of depth $d'-1$. By induction hypothesis, there must exist configurations $\mathcal{P}_l$, $\mathcal{P}_r$ in $\boldsymbol{\mathcal{P}}$ which contain $d'-1$ grey or black pebbles in $D_{d'-1,v_l}$ and $D_{d'-1,v_r}$, respectively, such that each path from a source to v_l/v_r contains at least one grey or black pebble. Let $\mathcal{P}_l$, $\mathcal{P}_r$ denote the last such configurations. Let $C^{(l)}_{d',v^*}$ denote the left-most path in $D_{d'}$, which passes through v_l.

Let's first consider the case that $\mathcal{P}_l = \mathcal{P}_r$. Since $C^{(l)}_{d',v^*}$ and $D_{d'-1,v_r}$ are disjoint, the properties of $\mathcal{P}_l$ and $\mathcal{P}_r$ imply that there must be at least d' grey or black pebbles in $\mathcal{P}_l = \mathcal{P}_r$ and since all paths to v^* either go through v_l or through v_r, also the second property is true and the claim follows with $\mathcal{P}^* := \mathcal{P}_l = \mathcal{P}_r$.

Now, w.l.o.g., assume $\mathcal{P}_l$ occurs before $\mathcal{P}_r$ in $\boldsymbol{\mathcal{P}}$. Thus, either 1) there are less than $d'-1$ pebbles on $D_{d'-1,v_l}$ in configuration $\mathcal{P}_r$ or 2) there must exist a path $C' \in D_{d'-1,v_l}$ from a source to v_l which does not carry any grey or black pebbles, in particular also one of v_l's parents is white-pebbled; and this is true for all configurations in $(\mathcal{P}_r, \ldots, \mathcal{P}_L)$. If case 2) does not occur, then similar to the case $\mathcal{P}_l = \mathcal{P}_r$ we can argue that there must be a grey or black pebble on $C^{(l)}_{d',v^*}$ and the claim follows. The same is true if node v^* is grey or black pebbled.

Finally, let's assume 2) is true for $\mathcal{P}_r$ and v^* is white-pebbled. Since v_l and v^* are W-pebbled, there must exist a configuration $\mathcal{P}' \in (\mathcal{P}_r, \ldots, \mathcal{P}_L)$ such that v_l is black-pebbled in $\mathcal{P}'$. Let $\mathcal{P}'$ be the first such configuration in $(\mathcal{P}_r, \ldots, \mathcal{P}_L)$. We will now construct a pebbling sequence on $D_{d'-1,v_l}$ which does *not* contain any configuration with at least $d'-1$ grey or black pebbles such that all paths to v_l carry at least one pebble – a contradiction to the induction hypothesis. Note that it suffices to show that the pebbling configuration on $D_{d'-1,v_l}$ induced by $\mathcal{P}_r$ can be reached by such a sequence, and then append this sequence by the pebbling strategy induced on $D_{d'-1,v_l}$ by $(\mathcal{P}_r, \ldots, \mathcal{P}')$. To define a pebbling strategy on $D_{d'-1,v_l}$ from the all-white

configuration to the one induced by $\mathcal{P}_r$, which always keeps one path all-white pebbled, we introduce some further notation: Let $C' = (u_1, \ldots, u_{d'-1})$ with $u_{d'-1} = v_l$ be represented as $(b_2, \ldots, b_{d-1}) \in \{0,1\}^{d'-2}$, where $b_i = 0/1$ indicates that u_{i-1} is the left/right parent of v_i. Furthermore, let $C^{(0)}_{i,u}/C^{(1)}_{i,u}$ denote the leftmost/rightmost path to node u on layer i. To define our pebbling strategy, we make the following simple observation: For any $i \in [2, d'-1]$, one can reach any configuration on $C^{(1-b_i)}_{i,u_i}$ with u_i white-pebbled while keeping the path $C^{(b_i)}_{i,u_i}$ all-$\mathtt{W}$ pebbled; this can be done by greedily black-pebbling all ancestors of grey or black pebbled nodes in $C^{(1-b_i)}_{i,u_i}$ and then reversibly switching all ancestors outside this set back to white (note, there are no ancestors in $C^{(b_i)}_{i,u_i}$, so this path remains white-pebbled). Thus, we define our pebbling strategy as follows

- For $i = d'-1, \ldots, 2$: Greedily black-pebble all ancestors of nodes in $C^{(1-b_i)}_{i,u_i}$ which are grey or black pebbled, and reach the pebbling configuration $\mathcal{P}_r$ induces on $C^{(1-b_i)}_{i,u_i}$, then reversibly white-pebble all ancestors of $C^{(1-b_i)}_{i,u_i}$.

Note, throughout the i-th step the path $C^{(b_i)}_{i,u_i} \cup (u_i, \ldots, u_{d'-1})$ remains white-pebbled and the pebbling configuration reached after the i-th step coincides with $\mathcal{P}_r$ on $\bigcup_{j\in[i,d'-1]} C^{(1-b_i)}_{i,u_i}$, and since u_i is white-pebbled in $\mathcal{P}_r$ the algorithm indeed terminates at the configuration which is induced on $D_{d'-1,v_l}$ by $\mathcal{P}_r$. This proves the claim. □

The following definition now gives a *cut in the configuration graph*; our adversary $\mathcal{A}$ will be a *threshold* adversary with respect to this cut.

Definition 74 (Good pebbling configurations)**.** A pebbling configuration $\mathcal{P}$ on DAG $\Gamma \setminus \Gamma^0$ is called *good* if it is *reachable* by reversible $\mathtt{WG}^3\mathtt{B}$ pebbling moves using less than d grey or black pebbles on the first d layers simultaneously, i.e., there exists a $\mathtt{WG}^3\mathtt{B}$ pebbling strategy $\boldsymbol{\mathcal{P}} := (\mathcal{P}_0, \ldots, \mathcal{P})$ for Γ such that $\sigma_{|\Gamma^{\oplus}}(\boldsymbol{\mathcal{P}}) \leq d-1$.

In particular, by Lemma 44, any pebbling configuration $\mathcal{P}$ with a $\mathtt{G}_L$ or $\mathtt{B}$ pebble on a gate in $\Gamma^{\wedge}$ is bad.

8.3.4 Extraction of Pebbling Configuration on $\Gamma \setminus \Gamma^0$

Given the garbled circuit $\widetilde{\Gamma}$ and input $\widetilde{x}$, our adversary $\mathcal{A}$ maps $(\widetilde{\Gamma}, \widetilde{x})$ to a *pebbling configuration* on the subgraph $\Gamma \setminus \Gamma^0$ of the DAG Γ underlying $\mathsf{\Gamma}$. It's output behaviour then depends on whether this pebbling configuration lies in the cut defined by Definition 74. In this section we will discuss how to extract such a pebbling configuration. Note, that $\mathcal{A}$ is computationally unbounded, hence can extract messages and keys from ciphertexts by brute-force search.

1. First, check whether $(\widetilde{\Gamma}, \widetilde{x})$ *evaluates correctly*, i.e., $\mathsf{GEval}(\widetilde{\Gamma}, \widetilde{x}) = \mathsf{\Gamma}(x_0)$.
   If the evaluation check passes, check whether $\widetilde{\Gamma}, \widetilde{x}$ have the *correct syntax*: Check whether $\widetilde{\Gamma}$ consists of four ciphertexts for each gate, which have the following form

$$\begin{aligned} \mathsf{c}_1 = \mathsf{Enc}_{\mathsf{k}_1}(\mathsf{Enc}_{\mathsf{k}_3}(\mathsf{k}_5)),\ \mathsf{c}_2 = \mathsf{Enc}_{\mathsf{k}_1}(\mathsf{Enc}_{\mathsf{k}_4}(\mathsf{m}_2)), \\ \{\mathsf{c}_3, \mathsf{c}_4\} = \{\mathsf{Enc}_{\mathsf{k}_2}(\mathsf{m}_3), \mathsf{Enc}_{\mathsf{k}_2}(\mathsf{m}_4)\}, \end{aligned} \tag{8.6}$$

for distinct keys $\mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{k}_4, \mathtt{k}_5$ and arbitrary (not necessarily distinct) messages $\mathtt{m}_2, \mathtt{m}_3, \mathtt{m}_4$, where keys $\mathtt{k}_1$ and $\mathtt{k}_3$ are revealed during evaluation $\mathsf{GEval}(\widetilde{\Gamma}, \widetilde{x})$. I.e., two of the four ciphertexts are encryptions under the same left secret keys $\mathtt{k}_1$ and $\mathtt{k}_2$, respectively, one of them is a double encryption $\mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5))$ under left key $\mathtt{k}_1$ and some right key $\mathtt{k}_3$ of an output key $\mathtt{k}_5$ (all these being revealed throughout evaluation), and the second encryption under $\mathtt{k}_1$ encrypts an encryption under a second right key $\mathtt{k}_4$ (of an arbitrary message $\mathtt{m}_2$).
Finally, check *consistency of keys*: For each gate, extract key pairs $(\mathtt{k}_1, \mathtt{k}_2)$ and $(\mathtt{k}_3, \mathtt{k}_4)$ corresponding to left and right input wires, and check whether they are consistent with the keys extracted from sibling gates: If gate g is the left sibling of g', then g's right input key pair must coincide with the left key pair extracted from g', i.e., $(\mathtt{k}_3, \mathtt{k}_4) = (\mathtt{k}'_1, \mathtt{k}'_2)$. Note, if this check passes, then all wires in the circuit can be uniquely associated with a key pair. Finally, check that all extracted keys are distinct.
If any of these checks fails, map $(\widetilde{\Gamma}, \widetilde{x})$ to a bad pebbling configuration, e.g., to the pebbling configuration on $\mathtt{G}$ where all gates at levels $[d+1, 2d+1]$ are black pebbled[11] and quit.

Remark 16. Note, syntax and consistency checks allow a reduction to distinguish

- a ciphertext from a non-ciphertext,
- a ciphertext under key $\mathtt{k}$ from a ciphertext under key $\mathtt{k}' \neq \mathtt{k}$.

We will argue in Section 8.3.5 that this is of no help to the reduction for breaking IND-CPA security of the information-theoretic encryption scheme $\mathcal{F}$.

For all garblings $(\widetilde{\Gamma}, \widetilde{x})$ that pass correctness, syntax, and consistency checks, $\mathcal{A}$ will extract a pebbling configuration on $\Gamma \setminus \Gamma^0$ by mapping each gate to a color in $\{\mathtt{W}, \mathtt{G}_*, \mathtt{G}_L, \mathtt{G}_R, \mathtt{B}\}$.

2. For each XOR gate g_j ($j \in [1,d] \cdot n + [0,n]$): Check whether g_j is garbled correctly with respect to input x_0. To this aim, let b_l, b_r, and $b_o = g_j(b_l, b_r) = b_l \oplus b_r$ denote the left/right input and the output bit of g_j, respectively, when evaluating Γ on x_0. We use the same notation as in Equation 8.6 above; furthermore, let $\mathtt{k}_6$ be the second key associated with the output wire (which was extracted from the garbling tables of the successor gates).

   - If g_j is garbled similar to the case of an honest garbling of (Γ, x_0), i.e., $\mathtt{m}_2 = \mathtt{k}_6$, $\mathtt{m}_3 = \mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_6)$, and $\mathtt{m}_4 = \mathsf{Enc}_{\mathtt{k}_4}(\mathtt{k}_5)$ (or the roles of $\mathtt{m}_3, \mathtt{m}_4$ permuted), then associate g_j with a $\mathtt{W}$ pebble.
   - If $\mathtt{m}_2$ and $\mathtt{m}_3$ are as in the previous case, but $\mathtt{m}_4 = \mathsf{Enc}_{\mathtt{k}_4}(\mathtt{m})$ for some message $\mathtt{m} \neq \mathtt{k}_5$, then associate g_j with a $\mathtt{G}_*$ pebble. Similarly for the case where the roles of $\mathtt{m}_3, \mathtt{m}_4$ are permuted.
   - If $\mathtt{m}_3$ is as in the first case, $\mathtt{m}_4 = \mathsf{Enc}_{\mathtt{k}_4}(\mathtt{m})$ for an arbitrary message $\mathtt{m}$, but $\mathtt{m}_2 \neq \mathtt{k}_6$, then associate g_j with a $\mathtt{G}_R$ pebble. Similarly for the case where the roles of $\mathtt{m}_3, \mathtt{m}_4$ are permuted.
   - If $\mathtt{m}_2 = \mathtt{k}_6$ is as in the first case, but $\{\mathtt{m}_3, \mathtt{m}_4\}$ differs from the previous cases, then associate g_j with a $\mathtt{G}_L$ pebble.
   - For all other cases, associate g_j with a $\mathtt{B}$ pebble.

[11] This choice was made for convenience (see Lemmas 46 to 48), but in principle could be an arbitrary bad configuration, and should simply guarantee that no reduction can gain any advantage by departing from the protocol in an obvious way.

Remark 17. Due to symmetry of the XOR operation, whether a gate is considered properly garbled (i.e. mapped to a white pebble) or not (i.e. mapped to grey or black) does *not* depend on the input keys. Thus, the set of black and grey pebbles on $\Gamma^{\oplus}$ can be extracted *independently* of x_0 and $\widetilde{x}$.

3. For each AND gate g_j ($j \in [d+1, 2d+1] \cdot n + [0, n]$): Similar to the case of XOR gates, check whether the gate is correctly garbled with respect to x_0. Using the same notation as above, associate g_j with a pebble as follows:

   - If g_j is garbled similar to the case of an honest garbling of (Γ, x_0), i.e., for
     $(b_l, b_r) = (0,0)$, we have $\mathtt{m}_2 = \mathtt{k}_5$, $\mathtt{m}_3 = \mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5)$, and $\mathtt{m}_4 = \mathsf{Enc}_{\mathtt{k}_4}(\mathtt{k}_6)$,
     $(b_l, b_r) = (0,1)$, we have $\mathtt{m}_2 = \mathtt{k}_5$, $\mathtt{m}_3 = \mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_6)$, and $\mathtt{m}_4 = \mathsf{Enc}_{\mathtt{k}_4}(\mathtt{k}_5)$,
     $(b_l, b_r) = (1,0)$, we have $\mathtt{m}_2 = \mathtt{k}_6$, $\mathtt{m}_3 = \mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5)$, and $\mathtt{m}_4 = \mathsf{Enc}_{\mathtt{k}_4}(\mathtt{k}_5)$,
     $(b_l, b_r) = (1,1)$, we have $\mathtt{m}_2 = \mathtt{k}_6$, $\mathtt{m}_3 = \mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_6)$, and $\mathtt{m}_4 = \mathsf{Enc}_{\mathtt{k}_4}(\mathtt{k}_6)$,
     (or the roles of $\mathtt{m}_3, \mathtt{m}_4$ permuted) then associate g_j with a $\mathtt{W}$ pebble.
   - If $\mathtt{m}_2$ and $\mathtt{m}_3$ are as in the previous case, but $\mathtt{m}_4 = \mathsf{Enc}_{\mathtt{k}_4}(\mathtt{m})$ for some message $\mathtt{m}$ that differs from above, then associate g_j with a $\mathtt{G}_*$ pebble. (Similarly for the case where the roles of $\mathtt{m}_3, \mathtt{m}_4$ are permuted.)
   - If $\mathtt{m}_3$ is as in the first case, $\mathtt{m}_4 = \mathsf{Enc}_{\mathtt{k}_4}(\mathtt{m})$ for an arbitrary message $\mathtt{m}$, but $\mathtt{m}_2$ differs from the previous case, then associate g_j with a $\mathtt{G}_R$ pebble. (Similarly for the case where the roles of $\mathtt{m}_3, \mathtt{m}_4$ are permuted.)
   - If $\mathtt{m}_2$ is as in the first case, but $\{\mathtt{m}_3, \mathtt{m}_4\}$ differs from the previous cases, then associate g_j with a $\mathtt{G}_L$ pebble.
   - For all other cases, associate g_j with a B pebble.

Remark 18. At first sight, it might seem counterintuitive that the mapping from gates to colours not only depends on the associated ciphertexts, but also on the input x_0. This however is unavoidable since the adversary $\mathcal{A}$ cannot simply map keys to bits, but can only *relate* them to the keys it learned from $\widetilde{x}$, which might be properly garbled or not.

In the following lemma, we prove that the adversary $\mathcal{A}$ using the above pebbling extraction indeed breaks indistinguishability of Yao's garbling scheme.

Lemma 45. *$\mathcal{A}$ breaks indistinguishability of the garbling scheme with probability $1 - 1/2^{n-1}$.*

Proof. We defined our adversary $\mathcal{A}$ to output $b' = 0$ whenever the extracted pebbling configuration is *good*, and $b' = 1$ else. In particular, Definition 74 guarantees that $\mathcal{A}$ outputs $b' = 0$ if there are only white pebbles on the subgraph $\Gamma \setminus \Gamma^0$ of the topology graph Γ of $\mathsf{\Gamma}$, i.e., when $(\widetilde{\mathsf{\Gamma}}_b, \widetilde{x}_b)$ is distributed identically to $(\widetilde{\mathsf{\Gamma}}, \widetilde{x}_0)$. On the other hand, when x_1 was garbled, then – as we will show below – there will be at least one grey or black pebble on layer $d+1$. Hence, since by Lemma 44 switching a pebble on layer $d' = d+1$ from $\mathtt{W}$ to $\mathtt{G}_*$, $\mathtt{G}_R$, $\mathtt{G}_L$ or $\mathtt{B}$ requires at least d grey or black pebles simultanously on the first d layers, $\mathcal{A}$ outputs $b' = 1$ in this case.

It remains to show that (with all but negligible probability) a proper garbling $(\widetilde{\mathsf{\Gamma}}, \widetilde{x}_1)$ will be mapped to a pebbling configuration which has at least one grey or black pebble on layer $d+1$. To this aim, we use the following properties of the circuit $\mathsf{\Gamma}^{\oplus}$, which were established in

Section 8.3.2, Corollary 19: First, $\Gamma^{\oplus}$ is 2-to-1, with x_0 and $x_0 \oplus 1^n$ being the two preimages of $\Gamma^{\oplus}(x_0)$. Second, the image of $\Gamma^{\oplus}$ only consists of strings containing an even number of 1s.

Now, assume $x_1 \notin \{x_0, x_0 \oplus 1^n\}$ (which happens with probability $1 - 1/2^{n-1}$), and $\Gamma^{\oplus}(x_0)$ and $\Gamma^{\oplus}(x_1)$ differ in the i-th bit and coincide in the $i+1$-th bit; note that such an i must exist, since $\Gamma^{\oplus}(x_0) \oplus 1^n$ is not in the image of $\Gamma^{\oplus}$ (by the second property of $\Gamma^{\oplus}$). Assume the i-th and $i+1$-th bits of $\Gamma^{\oplus}(x_0)$ are 0, i.e., using the same notation as above, we analyze the case $(b_l, b_r) = (0,0)$; the other cases work similarly. Consider the garbling of the i-th AND gate $\mathtt{G}$ on layer $d+1$ w.r.t. the keys $\mathtt{k}_1, \mathtt{k}_3, \mathtt{k}_5$ revealed through evaluation of $\widetilde{\Gamma}$ on input $\widetilde{x}_1$. Then this coincides with the case $(b_l, b_r) = (1,0)$, in particular, $\mathtt{m}_2 = \mathtt{k}_6$ and $\mathtt{m}_4 = \mathsf{Enc}_{\mathtt{k}_4}(\mathtt{k}_5)$ differ from the garbling for input x_0, while $\mathtt{m}_3$ is similar. Hence, $\mathcal{A}$ associates this gate with a $\mathtt{G}_R$ pebble. Similarly, we can see that if the left input coincides but the right differs, $\mathcal{A}$ associates $\mathtt{G}$ with a $\mathtt{G}_L$ pebble. Finally, if both inputs differ, this implies that in the garbling $\mathtt{m}_2$ and $\mathtt{m}_3$ differ, hence $\mathcal{A}$ maps $\mathtt{G}$ to a $\mathtt{B}$ pebble. This proves the claim. □

Since $\mathcal{A}$ extracts the pebble mode of a gate with regard to the garbled input (i.e., the keys it learns through evaluation), the reduction can still change the mode of a gate *after* it output $\widetilde{\Gamma}$ by choosing different input keys for $\widetilde{x}$. In the following lemmas we prove that this flexibility of choosing the input keys is of not much help to a reduction aiming at a good pebbling configuration, where in particular all gates at layers $[d+1, 2d+1]$ are mapped to $\mathtt{W}$, $\mathtt{G}_*$, or $\mathtt{G}_R$ pebbles.

First, we consider the case of a properly garbled AND gates. In this case, due to the asymmetry of the AND operation, input keys can be associated with bits and hence a properly garbled layer of AND gates has a similar function as an output mapping.

Lemma 46. *For any garbling of an AND gate on layer $[d+1, 2d+1]$, and any input bits b_l, b_r, there exists at most one input key pair (k_1, k_3) such that the gate will be mapped to a $\mathtt{W}$ pebble.*

Proof. For the claim on gates mapped to $\mathtt{W}$ pebbles, we only consider the case $(b_l, b_r) = (0,0)$, the others work similarly. Let $g_\wedge$ be an AND gate that is mapped to a $\mathtt{W}$ pebble. Hence, the four associated ciphertexts must have the following form

$$c_1 = \mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5)),\ c_2 = \mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{k}_5)),$$

$$c_3 = \mathsf{Enc}_{\mathtt{k}_2}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5)),\ c_4 = \mathsf{Enc}_{\mathtt{k}_2}(\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{k}_6)).$$

Since $g_\wedge$ is mapped to a $\mathtt{W}$ pebble, in particular evaluation, syntax, and consistency checks must pass. Hence, all keys are distinct, $\mathtt{k}_1, \mathtt{k}_2$ are associated to the left input wire, $\mathtt{k}_3, \mathtt{k}_4$ are associated to the right input wire, and during evaluation two input keys $\mathtt{k}_l \in \{\mathtt{k}_1, \mathtt{k}_2\}$ and $\mathtt{k}_r \in \{\mathtt{k}_3, \mathtt{k}_4\}$ are reveiled. We will now show that it must hold $(\mathtt{k}_l, \mathtt{k}_r) = (\mathtt{k}_1, \mathtt{k}_3)$. Assume, for contradiction, $(\mathtt{k}_l, \mathtt{k}_r) = (\mathtt{k}_2, \mathtt{k}_3)$. Then $g_\wedge$ will be mapped to a $\mathtt{G}_R$ pebble, because the inner encryptions (under key $\mathtt{k}_4$) of ciphertexts c_2 and c_4 are malformed. Similarly, if $(\mathtt{k}_l, \mathtt{k}_r) = (\mathtt{k}_1, \mathtt{k}_4)$, then c_3 and c_4 are considered malformed and $g_\wedge$ is mapped to a $\mathtt{G}_L$ pebble. Finally, if $(\mathtt{k}_l, \mathtt{k}_r) = (\mathtt{k}_2, \mathtt{k}_4)$, then c_2, and c_3 are considered malformed and $g_\wedge$ is mapped to a $\mathtt{B}$ pebble. This implies that $g_\wedge$ is mapped to a $\mathtt{W}$ pebble only if $(\mathtt{k}_l, \mathtt{k}_r) = (\mathtt{k}_1, \mathtt{k}_3)$. □

The situation becomes a bit more involved if AND gates are not properly garbled, since in this case asymmetry might be broken. However, if the left input keys can be mapped to bits, then we can still obtain some meaningful guarantees. We first consider the case that an AND

gate is garbled in $\mathtt{G}_*$ mode, i.e. one ciphertext is malformed and there exist some input bits (b_l, b_r) such that it will be mapped to a $\mathtt{G}_*$ pebble. In the following Lemma we prove that for a different right input bit $1 - b_r$ the gate will be mapped to a $\mathtt{G}_L$ pebble instead.

Lemma 47. *For any garbling of an AND gate, any left input bit b_l, and fixed left input key, there exists at most one $b_r \in \{0,1\}^*$ such that there exists a (not necessarily unique) right input key such that the gate will be mapped to a $\mathtt{G}_*$ pebble. If such a right input bit b_r exists, then for right input bit $1 - b_r$ the gate will be mapped to a $\mathtt{G}_L$ pebble.*

Proof. Consider the case that an AND gate $g_\wedge$ is mapped to a $\mathtt{G}_*$ pebble for $(b_l, b_r) = (0,0)$. In this case, the four ciphertexts associated to $g_\wedge$ must have the form

$$c_1 = \mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5)),\ c_2 = \mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{k}_5)),$$

$$c_3 = \mathsf{Enc}_{\mathtt{k}_2}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5)),\ c_4 = \mathsf{Enc}_{\mathtt{k}_2}(\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{m})),$$

for some message $\mathtt{m} \neq \mathtt{k}_6$ and left input key $\mathtt{k}_l = \mathtt{k}_1$. Now, for $(b_l, b_r) = (0,1)$ and $\mathtt{k}_l = \mathtt{k}_1$, a gate garbled as above will be mapped to a $\mathtt{G}_L$ pebble, no matter whether the right input key is $\mathtt{k}_3$ or $\mathtt{k}_4$.
Next, consider the case that an AND gate $g_\wedge$ is mapped to a $\mathtt{G}_*$ pebble for $(b_l, b_r) = (1,0)$. In this case, the four ciphertexts associated to $g_\wedge$ must have the form

$$c_1 = \mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5)),\ c_2 = \mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{k}_6)),$$

$$c_3 = \mathsf{Enc}_{\mathtt{k}_2}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5)),\ c_4 = \mathsf{Enc}_{\mathtt{k}_2}(\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{m})),$$

for some message $\mathtt{m} \neq \mathtt{k}_5$ and left input key $\mathtt{k}_l = \mathtt{k}_1$. Then, for $(b_l, b_r) = (1,1)$ and $\mathtt{k}_l = \mathtt{k}_1$, a gate garbled as above will be mapped to a $\mathtt{G}_L$ pebble, no matter whether the right input key is $\mathtt{k}_3$ or $\mathtt{k}_4$. □

Next we consider the case of an AND gate that is garbled in $\mathtt{G}_R$ mode w.r.t. some input bits (b_l, b_r). In this case we have to distinguish two different ways to garble a gate such that it will be mapped to a $\mathtt{G}_R$ pebble. For one type of $\mathtt{G}_R$ pebble we can map keys to bits, just as in the case of properly garbled gates. For the second type of $\mathtt{G}_R$ pebble we obtain a similar guarantee as for $\mathtt{G}_*$ pebbles.

Lemma 48. *For any garbling of an AND gate on layer $[d+1, 2d+1]$, any left input bit b_l, and fixed left input key, one of the following is true:*

1. *For any right input bit $b_r \in \{0,1\}^*$ there exists at most one right input key such that the gate will be mapped to a $\mathtt{G}_R$ pebble. If such a key exists, then for any other right input key the gate will be mapped to a $\mathtt{B}$ pebble.*
2. *There exists at most one input bit $b_r \in \{0,1\}^*$ such that there exists a right input key k_r such that the gate will be mapped to a $\mathtt{G}_R$ pebble. If such a bit exists, then for right input bit $1 - b_r$ and any right input key the gate will be mapped to a $\mathtt{B}$ pebble.*

These two cases characterize two different types of $\mathtt{G}_R$ pebbled gates, where we denote a gate as $\mathtt{G}_R$-type-1 if case 1 is true, and $\mathtt{G}_R$-type-2 if only *case 2 is true.*

Proof. First, consider the case that an AND gate $g_\wedge$ is mapped to a $\mathtt{G}_R$ pebble for $(b_l, b_r) = (0, 0)$. In this case, the four ciphertexts associated to $g_\wedge$ must have the form

$$c_1 = \mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5)),\ c_2 = \mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{m})),$$

$$c_3 = \mathsf{Enc}_{\mathtt{k}_2}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5)),\ c_4 = \mathsf{Enc}_{\mathtt{k}_2}(\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{m}')),$$

for some message $\mathtt{m} \neq \mathtt{k}_5$, an arbitrary message $\mathtt{m}'$, and left input key $\mathtt{k}_l = \mathtt{k}_1$. We first consider the case $\mathtt{m} = \mathtt{k}_6$, $\mathtt{m}' = \mathtt{k}_5$. Then, for $b_r = 0$ and right input key $\mathtt{k}_r = \mathtt{k}_4$, a gate garbled as above will be mapped to a $\mathtt{B}$ pebble; i.e. case 1 happens. Similarly, for $b_r = 1$ and right input key $\mathtt{k}_r = \mathtt{k}_3$, a gate garbled as above will be mapped to a $\mathtt{B}$ pebble; i.e. case 1 happens. Next consider the case $\mathtt{m} = \mathtt{k}_6$, $\mathtt{m}' = \mathtt{k}_6$. Then, for $b_r = 1$ and any right input key $\mathtt{k}_r$, a gate garbled as above will be mapped to a $\mathtt{B}$ pebble, i.e. case 2 happens. Finally, for $\mathtt{m}, \mathtt{m}' \notin \{\mathtt{k}_5, \mathtt{k}_6\}$, evaluation fails for $\mathtt{k}_r = \mathtt{k}_4$, hence the gate will be mapped to a $\mathtt{B}$ pebble. However, also for $b_r = 1$ the gate will be mapped to a $\mathtt{B}$ pebble; hence both cases 1 and 2 are true.

Next, consider the case that an AND gate $g_\wedge$ is mapped to a $\mathtt{G}_R$ pebble for $(b_l, b_r) = (1, 0)$. In this case, the four ciphertexts associated to $g_\wedge$ must have the form

$$c_1 = \mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5)),\ c_2 = \mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{m})),$$

$$c_3 = \mathsf{Enc}_{\mathtt{k}_2}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5)),\ c_4 = \mathsf{Enc}_{\mathtt{k}_2}(\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{m}')),$$

for some message $\mathtt{m} \neq \mathtt{k}_6$, an arbitrary message $\mathtt{m}'$, and left input key $\mathtt{k}_l = \mathtt{k}_1$. Then, for $\mathtt{m} = \mathtt{k}_5$, $\mathtt{m}' = \mathtt{k}_5$, and $b_r = 1$, the gate will be mapped to a $\mathtt{B}$ pebble, independently of the right input key, i.e. case 2 happens. For $\mathtt{m} = \mathtt{k}_5$, $\mathtt{m}' = \mathtt{k}_6$, on the other hand, if $b_r = 0$ and the right input key is $\mathtt{k}_r = \mathtt{k}_4$, then the gate will be mapped to a $\mathtt{B}$ pebble; and analogously, if $b_r = 1$ and the right input key is $\mathtt{k}_r = \mathtt{k}_3$, then the gate will be mapped to a $\mathtt{B}$ pebble, i.e. case 1 happens. For $\mathtt{m}, \mathtt{m}' \notin \{\mathtt{k}_5, \mathtt{k}_6\}$ the gate will be mapped to a $\mathtt{B}$ pebble and both cases are true.

Next, consider the case that an AND gate $g_\wedge$ is mapped to a $\mathtt{G}_R$ pebble for $(b_l, b_r) = (0, 1)$. In this case, the four ciphertexts associated to $g_\wedge$ must have the form

$$c_1 = \mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5)),\ c_2 = \mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{m})),$$

$$c_3 = \mathsf{Enc}_{\mathtt{k}_2}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_6)),\ c_4 = \mathsf{Enc}_{\mathtt{k}_2}(\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{m}')),$$

for some message $\mathtt{m} \neq \mathtt{k}_5$, an arbitrary message $\mathtt{m}'$, and left input key $\mathtt{k}_l = \mathtt{k}_1$. Then, for $\mathtt{m} = \mathtt{k}_6$, $\mathtt{m}' = \mathtt{k}_5$, and $b_r = 0$, the gate will be mapped to a $\mathtt{B}$ pebble, independently of the right input key, i.e. case 2 happens. For $\mathtt{m} = \mathtt{k}_6$, $\mathtt{m}' = \mathtt{k}_6$, on the other hand, if $b_r = 0$ and the right input key is $\mathtt{k}_r = \mathtt{k}_3$, then the gate will be mapped to a $\mathtt{B}$ pebble; and analogously, if $b_r = 1$ and the right input key is $\mathtt{k}_r = \mathtt{k}_4$, then the gate will be mapped to a $\mathtt{B}$ pebble, i.e. case 1 happens. For $\mathtt{m}, \mathtt{m}' \notin \{\mathtt{k}_5, \mathtt{k}_6\}$ the gate will be mapped to a $\mathtt{B}$ pebble and both cases are true.

Finally, consider the case that an AND gate $g_\wedge$ is mapped to a $\mathtt{G}_R$ pebble for $(b_l, b_r) = (1, 1)$. In this case, the four ciphertexts associated to $g_\wedge$ must have the form

$$c_1 = \mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_6)),\ c_2 = \mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{m})),$$

$$c_3 = \mathsf{Enc}_{\mathtt{k}_2}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5)),\ c_4 = \mathsf{Enc}_{\mathtt{k}_2}(\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{m}')),$$

for some message $\mathtt{m} \neq \mathtt{k}_5$, an arbitrary message $\mathtt{m}'$, and left input key $\mathtt{k}_l = \mathtt{k}_1$. Then, for $\mathtt{m} = \mathtt{k}_6$, $\mathtt{m}' = \mathtt{k}_5$, and $b_r = 0$, the gate will be mapped to a $\mathtt{B}$ pebble, independently of the right input key, i.e. case 2 happens. For $\mathtt{m} = \mathtt{k}_6$, $\mathtt{m}' = \mathtt{k}_6$, on the other hand, if $b_r = 0$ and the right input key is $\mathtt{k}_r = \mathtt{k}_3$, then the gate will be mapped to a $\mathtt{B}$ pebble; and analogously, if $b_r = 1$ and the right input key is $\mathtt{k}_r = \mathtt{k}_4$, then the gate will be mapped to a $\mathtt{B}$ pebble, i.e. case 1 happens. For $\mathtt{m}, \mathtt{m}' \notin \{\mathtt{k}_5, \mathtt{k}_6\}$ the gate will be mapped to a $\mathtt{B}$ pebble and both cases are true. □

8.3.5 Lower Bound on Security Loss for any Reduction

In this section we will combine all previous results to prove a lower bound on adaptive security of Yao's garbling scheme. More precisely, we will prove that any black-box reduction which aims to exploit $\mathcal{A}$'s distinguishing advantage to break IND-CPA security of the underlying encryption scheme loses a factor subexponential in the depth of the circuit.

Let R be an arbitrary PPT reduction which has black-box access to an adversary $\mathcal{A}$ that breaks indistinguishability of Yao's garbling scheme, and attempts to solve an IND-CPA challenge with respect to an encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Following the approach from Chapter 7, we define an *information-theoretically* secure encryption scheme $\mathcal{F} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as follows: For $l \in \{1, 6\}$, let $\mathsf{E}_l : \{0,1\}^{(l+2)\lambda} \to \{0,1\}^{2(l+2)\lambda}$ be a random expanding function (which is injective with overwhelming probability).

- Key generation $\mathsf{Gen}(1^\lambda)$: On input a security parameter λ in unary, output a key $\mathtt{k} \leftarrow \{0,1\}^*$ uniformly at random.
- Encryption $\mathsf{Enc}(\mathtt{k}, \mathtt{m})$: On input a key $\mathtt{k} \in \{0,1\}^\lambda$ and a message $\mathtt{m} \in \{0,1\}^{l \cdot \lambda}$ with $l \in \{1, 6\}$, sample randomness $r \leftarrow \{0,1\}^\lambda$, and output $\mathsf{E}_l(\mathtt{k}, \mathtt{m}; r)$.
- Decryption $\mathsf{Dec}(\mathtt{k}, \mathtt{c})$ is simulated to be consistent with Enc: On input a key $\mathtt{k} \in \{0,1\}^\lambda$ and a ciphertext $\mathtt{c} \in \{0,1\}^{2(l+2)\lambda}$ with $l \in \{1, 6\}$, check whether $\mathtt{c}$ lies in the image of $\mathsf{E}_l(\mathtt{k}, \cdot; \cdot)$, if so extract $\mathtt{m} \in \{0,1\}^{l \cdot \lambda}$, $r \in \{0,1\}^\lambda$ such that $\mathtt{c} = \mathsf{E}_l(\mathtt{k}, \mathtt{m}; r)$ and output $\mathtt{m}$, otherwise output $\perp$.

Choosing E_l ($l \in \{1, 6\}$) to be random functions implies that $\mathcal{F}$ is information-theoretically IND-CCA secure. Thus, since R only makes polynomially many queries, the only non-negligible advantage R has in breaking the IND-CPA security of $\mathcal{F}$ must stem from its interaction with $\mathcal{A}$. Furthermore, with all but negligible (in λ) probability $\mathcal{F}$ satisfies special correctness (Definition 45), hence can be used in Yao's garbling scheme.

We first argue that neither checking correctness, syntax, nor consistency (cf. Section 8.3.4) is of any help to R. Obviously, this is true for the correctness check, since R can efficiently evaluate $\mathsf{GEval}(\widetilde{\Gamma}, \widetilde{x})$. However, we have to argue a bit more to prove that also syntax and consistency checks are of no help to R. To this aim, we construct an oracle $\mathcal{O}$ that allows to distinguish

- a ciphertext from an arbitrary string in $\{0,1\}^{2(l+2)\lambda}$ for $l \in \{1, 6\}$,
- a ciphertext under key $\mathtt{k} \in \{0,1\}^\lambda$ from a ciphertext under key $\mathtt{k}' \neq \mathtt{k}$.

More precisely, $\mathcal{O}$ takes as input two strings $s \in \{0,1\}^{2(l+2)\lambda}$ and $s' \in \{0,1\}^{2(l'+2)\lambda}$ ($l, l' \in \{1, 6\}$) and checks whether s, s' lie in the image of $\mathsf{E}_l, \mathsf{E}_{l'}$, respectively. If this check fails for one of the strings, then $\mathcal{O}$ outputs $\perp$. Otherwise, it extracts preimages $(\mathtt{k}, \mathtt{m}, r) \in \{0,1\}^{(l+2)\lambda}$ under E_l and $(\mathtt{k}', \mathtt{m}', r') \in \{0,1\}^{(l'+2)\lambda}$ under $\mathsf{E}_{l'}$. If $\mathtt{k} = \mathtt{k}'$, $\mathcal{O}$ outputs 1, otherwise 0.

We will first prove that access to oracle $\mathcal{O}$ allows R to efficiently carry out syntax and consistency checks. Then we will prove that $\mathcal{F}$ remains information-theoretically IND-CPA secure even against adversaries that have access to $\mathcal{O}$.

Lemma 49. *There exists an algorithm* $\mathsf{B}^{\mathcal{F},\mathcal{O}}$ *with oracle access to* $\mathcal{O}$ *that given a garbled circuit and input pair* $(\widetilde{\Gamma}, \widetilde{x})$ *such that* $\mathsf{GEval}(\widetilde{\Gamma}, \widetilde{x}) = \Gamma(x_0)$ *efficiently checks whether* $(\widetilde{\Gamma}, \widetilde{x})$ *satisfies syntax and consistency (as defined in Section 8.3.4).*

Proof. We first describe how $\mathsf{B}^{\mathcal{F},\mathcal{O}}$ check the syntax of the four strings $s_1, s_2, s_3, s_4 \in \{0,1\}^{16\lambda}$ associated to a gate g. Since evaluation succeeds, one of these strings must have the form $\mathtt{c}_1 = \mathsf{Enc}_{\mathtt{k}_1}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_5))$ for three keys $\mathtt{k}_1, \mathtt{k}_3, \mathtt{k}_5 \in \{0,1\}^\lambda$ that are revealed during evaluation; w.l.o.g., assume $s_1 = \mathtt{c}_1$. Given $\mathtt{k}_1$, B can easily check if the strings s_2, s_3, s_4 can be decrypted under $\mathtt{k}_1$ by querying the decryption oracle Dec on $(\mathtt{k}_1, s_i)$ for all $i \in \{2,3,4\}$. The algorithm B aborts except if exactly on of the strings s_2, s_3, s_4 can be decrypted under $\mathtt{k}_1$, w.l.o.g., assume this string is s_2. Now given the decryption $\mathsf{Dec}(\mathtt{k}_1, s_2)$, B now checks whether this is a valid ciphertext by querying $\mathcal{O}(\mathsf{Dec}(\mathtt{k}_1, s_1), \mathsf{Dec}(\mathtt{k}_1, s_2))$. If the output of $\mathcal{O}$ is $\perp$, then $\mathsf{Dec}(\mathtt{k}_1, s_2)$ is not a valid ciphertext and B aborts. If the output is 1, then both s_1 and s_2 are double encryptions under the same key pair $(\mathtt{k}_1, \mathtt{k}_4) = (\mathtt{k}_1, \mathtt{k}_3)$, hence B aborts. Otherwise, B continues and queries $\mathcal{O}(s_3, s_4)$. B aborts, whenever the output to this query is not 1, i.e., s_3 and s_4 are not encryptions under the same key $\mathtt{k}_2$. Finally, B has to check that all keys used to generate s_1, s_2, s_3, s_4 are distinct. It already learned by previous queries that $\mathtt{k}_1 \neq \mathtt{k}_2$ and $\mathtt{k}_3 \neq \mathtt{k}_4$, hence it suffices to query $\mathcal{O}(\mathsf{Dec}(\mathtt{k}_1, s_i), s_j)$ for $i = 1, 2,\ j = 1, 3$ and abort if any output is not 0. Otherwise accept the syntax of $(\widetilde{\Gamma}, \widetilde{x})$.

Consistency of keys can be verified in a similar way using th oracle $\mathcal{O}$: Assume $(\mathtt{k}_1, \mathtt{k}_2), (\mathtt{k}_3, \mathtt{k}_4)$ are the (partially unknown) distinct keys involved in the encryptions $\mathtt{c}_1, \mathtt{c}_2, \mathtt{c}_3, \mathtt{c}_4$ (as guaranteed by the syntax check) associated to gate g; and similarly $(\mathtt{k}'_1, \mathtt{k}'_2), (\mathtt{k}'_3, \mathtt{k}'_4)$ and $\mathtt{c}'_1, \mathtt{c}'_2, \mathtt{c}'_3, \mathtt{c}'_4$ are the keys and ciphertexts associated with its right sibling g'. Since $\mathtt{k}_3$ and $\mathtt{k}'_1$ are revealed through evaluation, it is trivially true that $\mathtt{k}_3 = \mathtt{k}'_1$. For the unknown keys $\mathtt{k}_4$ and $\mathtt{k}'_2$, on the other hand, B can check equality using $\mathcal{O}$ on $(\mathsf{Dec}(\mathtt{k}_1, \mathtt{c}_2), \mathtt{c}'_3)$.

After checking that each wire in the circuit can uniquely be associated to a key pair, finally, B checks that all these keys are distinct: This works in a similar way as before by querying the oracle $\mathcal{O}$ on appropriate ciphertexts. □

Lemma 50. *The encryption scheme* $\mathcal{F}$ *is information-theoretically IND-CPA secure against adversaries with oracle access to* $\mathcal{O}$*.*

Proof. To see this, we show how the oracle $\mathcal{O}$ can be simulated by an IND-CPA challenger with oracle access to $\mathcal{F}$ that sees all the adversary's queries to $\mathcal{F}$. First, note that the probability of sampling a ciphertext from $\{0,1\}^{2(l+2)\lambda}$ ($l \in \{1,6\}$) *without* either calling Enc on a triple $(\mathtt{k}, \mathtt{m}, r) \in \{0,1\}^{(l+2)\lambda}$ or querying the IND-CPA oracle is negligible in λ. Thus, an oracle that allows to distinguish a ciphertext from an arbitrary string can be implemented simply by checking whether the queried string was output of a previous query; this is indistinguishable from the real functionality for any computationally bounded reduction. Furthermore, for all ciphertexts the reduction sees, it actually knows the corresponding keys, except for those derived from the IND-CPA challenger. Hence, distinguishing whether two given ciphertexts were derived under the same key is easy: In the case that both ciphertexts were derived from the IND-CPA challenger, obviously both ciphertexts were derived under the same key. In the case that at least one of the associated keys is known, the reduction can query the Dec oracle

on this key and the other ciphertext. Since Enc is injective with all-but-negligible probability, the answer to this query will be $\perp$ if the keys do not coincide. □

Now, to prove that any black-box reduction from indistinguishability of Yao's garbling scheme to IND-CPA security of the underlying encryption scheme suffers from a loss that is subexponential in the depth D of the circuit, we construct an adversary $\mathcal{A}[\mathtt{c}^*]$ that behaves just like $\mathcal{A}$ but doesn't decrypt challenge ciphertext $\mathtt{c}^*$. More precisely, $\mathcal{A}[\mathtt{c}^*]$ with input a ciphertext $\mathtt{c}^*$, has oracle access to $\mathcal{O}$, $\mathcal{F}$, as well as an IND-CCA decryption oracle $\mathsf{Dec}_{\mathtt{k}^*}$ that it can query on any ciphertext $c \neq \mathtt{c}^*$. We construct $\mathcal{A}[\mathtt{c}^*]$ such that it never decrypts $\mathtt{c}^*$ unless it already knows the encryption key $\mathtt{k}^*$ from other keys and ciphertexts in $\widetilde{\Gamma}, \widetilde{x}$:

- First $\mathcal{A}[\mathtt{c}^*]$ runs evaluation, syntax, and consistency checks using oracle $\mathcal{O}$. If these checks pass, similar to $\mathcal{A}$, the algorithm $\mathcal{A}[\mathtt{c}^*]$ uses brute-force search to decrypt all ciphertexts *except* for those encrypted under $\mathtt{k}^*$ (to check whether a ciphertext is encrypted under $\mathtt{k}^*$ it uses $\mathcal{O}$ and $\mathtt{c}^*$). Ciphertexts $\mathtt{c} \neq \mathtt{c}^*$ encrypted under $\mathtt{k}^*$ it decrypts using oracle $\mathsf{Dec}_{\mathtt{k}^*}$. For $\mathtt{c}^*$, there are two cases:
  - If the key $\mathtt{k}^*$ was learned from previous decryptions (this can be checked by decrypting $\mathtt{c}^*$ under all known keys), $\mathcal{A}[\mathtt{c}^*]$ simply decrypts $\mathtt{c}^*$ using $\mathtt{k}^*$.
  - If the $\mathtt{k}^*$ is not known to $\mathcal{A}[\mathtt{c}^*]$, then it simply *assumes* $\mathtt{c}^* \in \{0,1\}^{2(l+2)\lambda}$ with $l \in \{1,6\}$ would decrypt to $0^{l\cdot\lambda}$.

  $\mathcal{A}[\mathtt{c}^*]$ then continues analogous to $\mathcal{A}$ by mapping $(\widetilde{\Gamma}, \widetilde{x})$ to a pebbling configuration and outputting 0 whenever the pebbling configuration is good per Definition 74, and 1 otherwise.

Clearly, since $\mathcal{A}[\mathtt{c}^*]$ never decrypts $\mathtt{c}^*$ except if $\mathtt{k}^*$ is known, there is no chance for R to use $\mathcal{A}[\mathtt{c}^*]$ to break IND-CPA security of $\mathcal{F}$.[12] It remains to bound the success probability of any PPT distinguisher D to distinguish $\mathcal{A}[\mathtt{c}^*]$ from $\mathcal{A}$.[13] To this aim, we will first show how the $\mathtt{WG}^3\mathtt{B}$ pebbling game relates to this issue.

Lemma 51. *Let $c^* \leftarrow \mathsf{Enc}_{k^*}(m)$ be an arbitrary ciphertext and let $\mathcal{P}$, $\mathcal{P}^*$ be the two pebbling configurations extracted by $\mathcal{A}$ and $\mathcal{A}[\mathtt{c}^*]$, respectively, in the same execution of the game, i.e. using the same randomness. Then $\mathcal{P}^*$ differs from $\mathcal{P}$ by at most one valid $\mathtt{WG}^3\mathtt{B}$ pebbling move.*

Proof. First, note that whenever $\mathtt{c}^*$ is not embedded into $\widetilde{\Gamma}$ then $\mathcal{A}[\mathtt{c}^*]$ is trivially indistinguishable from $\mathcal{A}$, as $\mathcal{A}[\mathtt{c}^*] \equiv \mathcal{A}$ in this case. Similarly, if $\mathtt{k}^* \in \widetilde{x}$ or there exists any encryption of $\mathtt{k}^*$ in $\widetilde{\Gamma}$, then also $\mathcal{A}[\mathtt{c}^*] \equiv \mathcal{A}$; in particular, $\mathcal{P} = \mathcal{P}^*$. Now, assume that evaluation fails for $\mathcal{A}[\mathtt{c}^*]$ but passes in $\mathcal{A}$. Then the key $\mathtt{k}^*$ must be revealed through $\mathsf{GEval}(\widetilde{\Gamma}, \widetilde{x})$. But then $\mathcal{A}[\mathtt{c}^*]$ properly decrypts $\mathtt{c}^*$; hence this cannot happen and evaluation fails for $\mathcal{A}[\mathtt{c}^*]$ if and only it fails for $\mathcal{A}$. Also syntax and consistency checks $\mathcal{A}[\mathtt{c}^*]$ passes if and only if $\mathcal{A}$ passes. Thus, whenever such an initial check fails, then $\mathcal{P} = \mathcal{P}^*$.

In the following we will assume that $\mathtt{c}^*$ is embedded in $\widetilde{\Gamma}$, the key $\mathtt{k}^*$ is never encrypted or opened in $\widetilde{x}$, and all initial checks pass. The second assumption implies that the key $\mathtt{k}^*$

[12] Recall that our ideal encryption scheme $\mathcal{F}$ is IND-CCA secure, hence access to the oracle $\mathsf{Dec}_{\mathtt{k}^*}$ used by $\mathcal{A}[\mathtt{c}^*]$ is of no help to R.

[13] Note, we assume that $\mathcal{A}[\mathtt{c}^*]$ has *private* access to its oracles and D cannot observe its oracle queries to distinguish it from $\mathcal{A}$.

must either be embedded at a non-opened input wire or at the output wire of a gate whose associated ciphertexts only encode one of the two output keys. We will now argue that such XOR gates will be mapped to $\mathtt{B}$ pebbles: Using the notation from Section 8.3.4, the key $\mathtt{k}_5$ is learned during evaluation, while $\mathtt{k}_6$ is the second key associated to the output wire. If the garbling of the XOR gate is independent of $\mathtt{k}_6$, this implies that messages $\mathtt{m}_2$ and $\mathtt{m}_3$ differ from the case of an honest garbling. By definition, such gates are mapped to $\mathtt{B}$ pebbles. Hence, $\mathtt{c}^*$ can only be embedded in the garbling table of a gate g if at least one of g's parents is black pebbled.

Now, assume there exists a (XOR or AND) gate that is $\mathtt{W}$ in $\mathcal{P}$ and $\mathtt{B}$ in $\mathcal{P}^*$; the case opposite case works analogously. For ease of notation, we consider the case of an XOR gate here, the case of AND gates follows analogously. By definition, an XOR gate is only mapped to a $\mathtt{B}$ gate, if either (1) $\mathtt{m}_2 \neq \mathtt{k}_6$ and $\mathtt{m}_3 \notin \mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_6)$, or (2) $\mathtt{m}_2 \neq \mathtt{k}_6$ and $\mathtt{m}_4 \notin \mathsf{Im}(\mathsf{Enc}_{\mathtt{k}_4})$. For case (1), note that since $\mathtt{k}_3$ is known, $\mathtt{m}_2$ and $\mathtt{m}_3$ can only be switched by embedding $\mathtt{c}^*$ *twice*, for right key $\mathtt{k}_4$ *and* left key $\mathtt{k}_2$. The same is true for case (2). But since by consistency $\mathtt{k}_2 \neq \mathtt{k}_4$, this implies that $\mathcal{P}$ and $\mathcal{P}^*$ cannot differ by a switch from $\mathtt{W}$ to B.

Next, consider the case that a gate is mapped to $\mathtt{W}$ in $\mathcal{P}$ and to $\mathtt{G}_L$ in $\mathcal{P}^*$. We will argue that in this case the key $\mathtt{k}^*$ must be embedded at the left input wire. Again, for ease of notation, we consider the case of an XOR gate g; the case of AND gates follows analogously. Since $\mathcal{A}$ maps g to $\mathtt{W}$ and $\mathcal{A}[\mathtt{c}^*]$ maps it to $\mathtt{G}_L$, g must be properly garbled and the change arrises from $\mathcal{A}[\mathtt{c}^*]$ "decrypting" $\mathtt{c}^*$ to 0. Hence, $\mathtt{c}^*$ must be embedded either at $\mathtt{c}_3$ or $\mathtt{c}_4$, in particular $\mathtt{k}^* = \mathtt{k}_2$ the *left* input key. Furthermore, since g is properly garbled, it must hold either $\mathtt{c}^* \leftarrow \mathsf{Enc}_{\mathtt{k}^*}(\mathsf{Enc}_{\mathtt{k}_3}(\mathtt{k}_6))$ or $\mathtt{c}^* \leftarrow \mathsf{Enc}_{\mathtt{k}^*}(\mathsf{Enc}_{\mathtt{k}_4}(\mathtt{k}_5))$, i.e. $\mathtt{c}^*$ can only be embedded *once* in this gate. If $\mathtt{c}^*$ was additionally embedded in another gate, then this must be the left sibling g' of g due to consistency and distinctness of keys. In g', however, $\mathtt{k}^*$ is employed as a right input key, hence $\mathtt{c}^*$ can only be embedded as an inner encryption and in particular constitutes a malformed encryption (note the difference in length between inner and outer encryptions). Thus, both $\mathcal{A}$ and $\mathcal{A}[\mathtt{c}^*]$ will map g' to the same pebble.

In a similar way, one can prove that whenever a gate is mapped to $\mathtt{W}$ in $\mathcal{P}$ and to $\mathtt{G}_R$ in $\mathcal{P}^*$, then the key $\mathtt{k}^*$ must be embedded at the right input wire. Also in this case it follows that $\mathtt{c}^*$ can be embedded in at most one further gate – the right sibling – and there will be considered as a malformed ciphertext by both $\mathcal{A}$ and $\mathcal{A}[\mathtt{c}^*]$.

For the case that a gate is mapped to $\mathtt{W}$ in $\mathcal{P}$ and to $\mathtt{G}_*$ in $\mathcal{P}^*$, the key $\mathtt{k}^*$ can be embedded either at the right or the left input wire – however (by consistency) not at both. Also in this case it follows that $\mathtt{c}^*$ can be embedded in at most one further gate, and for this other gate $\mathtt{c}^*$ will be considered as a malformed ciphertext by both $\mathcal{A}$ and $\mathcal{A}[\mathtt{c}^*]$.

Analogously, one can verify the remaining $\mathtt{WG}^3\mathtt{B}$ pebbling rules by analyzing the cases of a gate being mapped to $\mathtt{G}_*$ ($\mathtt{G}_L/\mathtt{G}_R$) in $\mathcal{P}$ and to $\mathtt{G}_L/\mathtt{G}_R$ ($\mathtt{B}$) in $\mathcal{P}^*$. Also in these cases, embedding $\mathtt{c}^*$ at any further gate leads to $\mathcal{A}$ and $\mathcal{A}[\mathtt{c}^*]$ extracting the same pebble for this further gate. □

We will now bound the distinguishing advantage of $\mathsf{D}^{\mathcal{F}}$. Recall that a pebbling configuration on $\Gamma \setminus \Gamma^0$ is good per Definition 74 if it can be reached by $\mathtt{WG}^3\mathtt{B}$ pebbling moves using at most $d-1$ pebbles on the first d layers. Thus, by Lemma 51, any successful distinguisher D has to simulate $\widetilde{\Gamma}$ and $\widetilde{x}$ such that the pebbling configurations $\mathcal{P}, \mathcal{P}^*$ on Γ extracted by $\mathcal{A}$ and $\mathcal{A}[\mathtt{c}^*]$, respectively, contain exactly $d-1$ or d black and grey pebbles on the first d layers (depending on the IND-CPA challenge bit b^*), contain only $\mathtt{W}$, $\mathtt{G}_*$, and $\mathtt{G}_R$ pebbles on higher layers, and differ by a valid $\mathtt{WG}^3\mathtt{B}$ pebbling move within layers $[1, d+1]$.

In the following we will first restrict our analysis to *non-rewinding* distinguishers and assume x_0, x_1 were chosen uniformly at random by $\mathcal{A}$ *after* it sees $\widetilde{\Gamma}$. Finally we will discuss how to slightly modify our adversary $\mathcal{A}$ to also cover the case that D chooses $\mathcal{A}$'s randomness and rewinds $\mathcal{A}$.

To bound the success probability of D, let r be arbitrary random coins and consider two cases:

(1) there exists s such that the output of $\mathcal{A}(s)$ and $\mathcal{A}[\mathsf{C}^*](s)$ after interaction with $\mathsf{D}(r, \mathsf{C}^*)$ differs and in $\mathcal{P}$ and $\mathcal{P}^*$ there are *more than* $\bar{d}$ many $\mathtt{G}_*$ and $\mathtt{G}_R$-type-2 (as defined in Lemma 48) pebbles in layers $[d+2, 2d+1]$,

(2) there exists s such that the output of $\mathcal{A}(s)$ and $\mathcal{A}[\mathsf{C}^*](s)$ after interaction with $\mathsf{D}(r, \mathsf{C}^*)$ differs and in $\mathcal{P}$ and $\mathcal{P}^*$ there are *at most* $\bar{d}$ many $\mathtt{G}_*$ and $\mathtt{G}_R$-type-2 pebbles in layers $[d+2, 2d+1]$.

We leave the parameter $\bar{d} < d/3$ undefined for now and optimze it later. In Lemmas 52 and 53, we will argue that, intuitively, in both cases the distinguisher D must have correctly guessed many of the input bits in x_0.

Lemma 52. *Let r be arbitrary coins such that case (1) is true. Then the probability (over uniformly random coins s) that the output of $\mathcal{A}(s)$ and $\mathcal{A}[\mathsf{C}^*](s)$ differs after interaction with $\mathsf{D}(r, \mathsf{C}^*)$ is at most $(3/4)^{\sqrt{\bar{d}}/7}$.*

Proof. To prove this lemma, we will use Lemmas 46 to 48. First, note that D can only succeed if at most one of the gates at layer $d+1$ is not mapped to a $\mathtt{W}$ pebble, since the adversary $\mathcal{A}$ outputs 1 whenever any gate at layer $d+1$ is not $\mathtt{W}$ pebbled. Now, by Lemma 46, there is at most one pair of input keys to an AND gate that leads to this gate being mapped to a $\mathtt{W}$ pebble. As the input to all but one gate at layer $d+1$ comprises *all* input to layer $d+1$, this implies that D can only succeed, if it properly garbles all gates at layer $d+1$ and the input keys which are revealed through $\mathsf{GEval}(\widetilde{\Gamma}, \widetilde{x})$ are associated with the corresponding bits in $\Gamma^{\oplus}(x_0)$.

Next, consider the AND gates at layers $[d+2, 2d+1]$. For D to succeed, these gates must *not* end up $\mathtt{G}_L$ or $\mathtt{B}$ pebbled. Since all these gates have their left input from layer d and by the previous argument all these keys are fixed, we can apply Lemmas 47 and 48: Let $\mathcal{S}$ denote the set of $\bar{d}$ gates in layers $[d+2, 2d+1]$ that are mapped to $\mathtt{G}_*$ or $\mathtt{G}_R$-type-2 pebbles (for some random coins s such that (1) is true). Then by Lemma 43 there exists a subset $\mathcal{S}' \subseteq \mathcal{S}$ of size $\sqrt{\bar{d}}/4$ such that the set of right parents $\mathcal{S}_R$ of $\mathcal{S}'$ is linearly independent over $\mathbb{Z}_2$; and for each gate $g \in \mathcal{S}'$ left and right parent are linearly independent. To see that the latter is true, note that any subset smaller than n of gates within one layer or within one column is linearly independent (cf. Lemma 42). It directly follows that left and right parents of any gate $g \in \mathcal{S}'$ since they lie in the same column. Furthermore, the set of left parents $\mathcal{S}_L$ to $\mathcal{S}'$ is linearly independent since it is a subset of $\leq \bar{d} < n$ gates at layer d.

To argue that D must have guessed many of the right input bits to $S^{\wedge}$ correctly, we use the following simple result from linear algebra.

Claim 4. *Let $m \in [1, n]$ and $\mathcal{S}_1 = \{u_i\}_{i \in [1,m]}$ a subset of $\{0,1\}^n$ that is linearly independent over $\mathbb{Z}_2$. Let $\mathcal{S}_2 = \{v_i\}_{i \in [1,m]}$ be a multiset of elements in $\{0,1\}^n$ such that $\mathcal{S}_2$ as a set is linearly independent over $\mathbb{Z}_2$. Furthermore, assume $\{u_i, v_i\}$ is linearly independent for all $i \in [1, m]$. Then there exists an index set $\mathcal{I} \subset [1, m]$ of size $|\mathcal{I}| = \lfloor m/4 \rfloor$ such that $\bigcup_{i \in \mathcal{I}} \{u_i\} \cup \{v_i\}$ is linearly independent.*

Proof of the claim. Let $i_1 \in [1,m]$ be arbitrary and set $\mathcal{I}_1 := \{i_1\}$. Then clearly $|\mathcal{I}_1| = 1$ and $\mathcal{U}_1 := \bigcup_{j\in\mathcal{I}_1}\{u_j\} \cup \{v_j\}$ is linearly independent. For $k > 1$, choose $i_k \in [1,m] \setminus \mathcal{I}_{k-1}$ such that $\mathcal{U}_k := \mathcal{U}_{k-1} \cup \{u_{i_k}\} \cup \{v_{i_k}\}$ is linearly independent, if exists, otherwise set $i_k := i_{k-1}$; set $\mathcal{I}_k := \mathcal{I}_{k-1} \cup \{i_k\}$. We show that such $i_k \in [1,m] \setminus \mathcal{I}_{k-1}$ must exist as long as $k \leq m/4$: Since $\dim(\mathcal{U}_{k-1}) \leq 2(k-1) \leq m/2 - 2$ and $\mathcal{S}_1$ is linearly independent, it must hold $|\mathcal{S}_1 \cap \langle\mathcal{U}_{k-1}\rangle| \leq m/2 - 2$; we denote the index set of $\mathcal{S}_1 \cap \langle\mathcal{U}_{k-1}\rangle$ by $\mathcal{I}_{\mathcal{S}_1,k-1}$. For $\mathcal{S}_2$, on the other hand, it also holds that $|\mathcal{S}_2 \cap \langle\mathcal{U}_{k-1}\rangle| \leq m/2 - 2$, but since $\mathcal{S}_2$ is a *multiset* we don't obtain a lower bound on $|\mathcal{S}_2 \setminus \langle\mathcal{U}_{k-1}\rangle|$. However, since for any linearly independent set $\mathcal{U}$ and $v \in \mathcal{U}$ the set $\mathcal{U} \cup \{v\} = U$ is linearly independent, we choose $\mathcal{I}_{\mathcal{S}_2,k-1} \subset [1,m]$ minimal such that $\bigcup_{j\in\mathcal{I}_{\mathcal{S}_2,k-1}}\{u_j\} = \mathcal{S}_2 \cap \langle\mathcal{U}_{k-1}\rangle$. Then again we have $|\mathcal{I}_{\mathcal{S}_2,k-1}| \leq m/2 - 2$. Thus, by pigeonhole principle, there must exist $i_k \in [1,m] \setminus (\mathcal{I}_{\mathcal{S}_1,k-1} \cup \mathcal{I}_{\mathcal{S}_2,k-1})$ such that $\mathcal{U}_k := \mathcal{U}_{k-1} \cup \{u_{i_k}\} \cup \{v_{i_k}\}$ is linearly independent and we have $|\mathcal{I}_k| = |\mathcal{I}_{k-1} \cup \{i_k\}| = k$. □

Since the multiset $\mathcal{S}_L$ and the set $\mathcal{S}_R$ of left and right parents of $\mathcal{S}'$ are linearly independent (as sets), respectively, and for any $g \in \mathcal{S}'$ left and right input to $\mathtt{G}$ are linearly independent, we can apply the claim to obtain a subset $\mathcal{S}'' \subset \mathcal{S}'$ of size $|\mathcal{S}'|/4$ such that the union of the parents of $\mathcal{S}''$ is linearly independent. For $\mathcal{S}''$, we can now use Lemmas 47 and 48 to see that any successful D must have correctly guessed all right input bits to $\mathcal{S}''$; i.e., for s sampled uniformly at random, the probability that D succeeds is at most $(1/2)^{|\mathcal{S}''|}$. As $|\mathcal{S}'| \geq \sqrt{\bar{d}}/4$, the probability that D succeeds can be upper-bounded by

$$\Pr[\text{D succeeds in case (1)}] \leq \left(\frac{1}{2}\right)^{\sqrt{\bar{d}}/16} < \left(\frac{3}{4}\right)^{\sqrt{\bar{d}}/7}.$$

□

Lemma 53. *Let r be arbitrary coins such that case (2) is true. Then the probability (over uniformly random coins s) that the output of $\mathcal{A}(s)$ and $\mathcal{A}[c^*](s)$ differs after interaction with $\mathsf{D}(r, c^*)$ is at most $(3/4)^{\sqrt{d-3\bar{d}}/4}$.*

Proof. Recall that whenever the consistency check passes, each wire in $\widetilde{\Gamma}$ can be uniquely associated with two keys. Now, in case (2), for all but $\bar{d}$ wires in $\Gamma \setminus \Gamma^0$ the following holds: By Lemmas 46 and 48, for each bit running over the wire w in Γ, there exists at most one key associated with w in $\widetilde{\Gamma}^{\oplus}$ such that the AND gates with right input wire w is mapped to a "good" (W or $\mathtt{G}_R$-type-1) pebble, while for the other key associated to w it would be mapped to a "bad" pebble ($\mathtt{G}_L$ or $\mathtt{B}$). Note that in the latter case D immediately fails.

This allows us to map keys associated with wires in $\widetilde{\Gamma}^{\oplus}$ to bits, hence implies a mapping from $(\widetilde{\Gamma}, \widetilde{x})$ to a circuit $\hat{\Gamma}$ and input $\hat{x}$, where $\hat{\Gamma}$ contains at most $3\bar{d}$ "undefined" gates (note, each internal wire effects 3 gates in $\Gamma^{\oplus}$). Now, for D to succeed, it has to simulate $(\widetilde{\Gamma}, \widetilde{x})$ such that at least $d' := d - 3\bar{d}$ "well-defined" gates in the circuit $\hat{\Gamma}$ differ from XOR gates and $\hat{x} = x_0$. At the same time, all input and output wires of the well-defined gates have to carry the correct bits during evaluation (for "evaluation" of $\hat{\Gamma}$ on $\hat{x}$ we apply the mapping from keys to bits to $\mathsf{Eval}(\widetilde{\Gamma}, \widetilde{x})$ to extract a bit for all wires connected to well-defined gates).

Ignoring the undefined gates in $\hat{\Gamma}$, this exactly corresponds to the game introduced in Section 8.3.2: D simulates a circuit such that all but d' gates are garbled correctly as XOR gates, and D succeeds, if for all gates the (input and) output bits correspond to the respective bits during evaluation of $\Gamma^{\oplus}$ on input x_0. Lemma 41 now implies an upper bound on D's success

probability in case (2):

$$\Pr[\text{D succeeds in case (2)}] \leq \left(\frac{3}{4}\right)^{\sqrt{d'}/4} = \left(\frac{3}{4}\right)^{\sqrt{d-3\bar{d}}/4}.$$

□

Thus, Lemmas 52 and 53 imply the following bound on any non-rewinding PPT distinguisher D (choose $\bar{d} = d/4$):

Corollary 20. *No non-rewinding PPT distinguisher* $\mathsf{D}^{\mathcal{F}}$ *can distinguish* $\mathcal{A}[\mathsf{c}^*]$ *from* $\mathcal{A}$ *with probability larger than* $(3/4)^{\sqrt{d}/14}$.

To handle arbitrary – potentially rewinding – distinguishers D, we modify $\mathcal{A}$ as follows: Instead of sampling x_0, x_1 using random coins s, we assume a pseudorandom function f_k with uniformly random key k was hardcoded in $\mathcal{A}$, which takes as input a garbled circuit $\widetilde{\Gamma}$ and coins s, and outputs a tuple (x_0, x_1). Since D only has *black-box* access to $\mathcal{A}/\mathcal{A}[\mathsf{c}^*]$, the secret key k is hidden from D, thus for two different inputs $(\widetilde{\Gamma}, s), (\widetilde{\Gamma}', s')$ to $\mathcal{A}/\mathcal{A}[\mathsf{c}^*]$ the input pairs $(x_0, x_1), (x_0', x_1')$ look like independently sampled uniformly random strings.

With this modification in place, we finally arrive at the following lower bound on the security loss of any *black-box reduction* R (where we used $D < 3d$, hence $\sqrt{d}/14 > \sqrt{D}/25$). Note that our bounds naturally only apply to $d \leq n$, hence we assume $D < 2n$ in our theorem statement.

Theorem 32. *Any black-box reduction from the indistinguishability of Yao's garbling scheme (or its variant from [JW16]) on the class of circuits with input length* n *and depth* $D \leq 2n$ *to the IND-CPA security of the underlying encryption scheme loses at least a factor* $\frac{1}{q} \cdot (\frac{4}{3})^{\sqrt{D}/25} > \frac{1}{q} \cdot 2^{\sqrt{D}/61}$, *where* q *denotes the number of times the reduction rewinds the adversary.*

8.4 Conclusion and Open Problems

In this work we prove that any blackbox reduction from indistinguishability of (the modification [JW16] of) Yao's garbling scheme to IND-CPA security of the underlying encryption scheme must involve a loss in security that is subexponential in the depth of the circuit. This clearly also implies limitations to the stronger and more common simulation-based security and shows that the approach of [JW16] is essentially optimal. However, we leave it to future work if our finegrained separation can be turned into an actual attack against Yao's garbling scheme.

Beside this most exciting open problem, one can also consider if our approach can be optimized. It might be possible to push our lower bound to an exponential loss, which would exactly match the upper bound from [JW16]. Following our approach, this requires a more sophisticated pebbling lower bound. Another interesting question would be if an even stronger bound can be found for the original construction of Yao, where the output mapping is sent in the offline phase, and certain limitations are already known from [AIKW13].

Part V

Conclusion

CHAPTER 9

Conclusion

In this thesis we have explored the problem of proving adaptive security, both from a positive and a negative direction. We started by analyzing several existing reductions for generalized selective decryption (GSD) [Pan07, FJP15], the [GGM84] construction of constrained psudorandom functions (cPRF) [FKPR14], and a natural variant of Yao's garbling scheme [JW16]. We extracted out the underlying ideas these works have in common by providing a framework for proving adaptive security (Chapter 3). This *Piecewise-Guessing framework* allowed us to present the former results in a unified and simplified way.

We then applied our framework to several constructions of other cryptographic primitives for which previously only selective security had been proven: various proxy re-encryption schemes (PRE, Chapter 4), the TreeKEM [BBR18] protocol for continuous group key agreement (CGKA, Chapter 5), and Yao's (original) garbling scheme [Yao82, Yao86] (Chapter 6). While in some settings we indeed achieved adaptive security at only a (quasi-)polynomial loss in security, we also encountered limitations of the techniques provided by the framework.

These limitations we explored in the last part of this thesis, where we showed that currently used techniques cannot lead to significantly better bounds on adaptive security: First (Chapter 7), we considered *multi-round* security games such as GSD and PRE, where an adversary queries edges of some graph and the difficulty for the reduction lies in guessing where these edges will end up in the final graph. We note that all known reductions for such games are *oblivious* to the adversary's behaviour, i.e. they do not make use of the partial information they learn during the game. While obliviousness obviously is a very strong restriction to a reduction and we obtain significantly stronger lower bounds for oblivious reductions, it remains an open problem whether non-obliviousness can be exploited to obtain better upper bounds. Another (though related) restriction common to all known reductions is that they are *non-rewinding*. Most of our lower bounds only hold for non-rewinding reductions and rewinding seems to be a very strong tool in such multi-round games. However, it is not at all clear at this point how this tool could be exploited towards improved upper bounds.

The picture is significantly clearer in the setting of Yao's garbling scheme (Chapter 8), where the entire circuit is sent in one step and the difficulty for the reduction lies in guessing (parts of) the input. Here we proved a lower bound almost matching the known upper bound, which holds for arbitrary (potentially rewinding) black-box reductions. While this result indicates some weakness of Yao's garbling, it does not provide an attack against the scheme. We leave it as an exciting open problem if our lower bound can be turned into a concrete counter example,

i.e. an *efficient* IND-CPA secure encryption scheme that allows the attacker to *efficiently* break adaptive security of the scheme. We expect this to require rather strong additional assumptions.

While we leave many interesting questions open, we hope that this thesis still contributes to a better understanding of the difficulty in proving adaptive security and will inspire future research in this area.

Bibliography

[AAB+21] Joel Alwen, Benedikt Auerbach, Mirza Ahad Baig, Miguel Cueto-Noval, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, and Michael Walter. Grafting key-trees: Efficient key-management for overlapping groups. TCC 2021, to appear, 2021.

[ABH09] Giuseppe Ateniese, Karyn Benson, and Susan Hohenberger. Key-private proxy re-encryption. In Marc Fischlin, editor, *Topics in Cryptology – CT-RSA 2009*, pages 279–294, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[ACC+16] Prabhanjan Ananth, Yu-Chi Chen, Kai-Min Chung, Huijia Lin, and Wei-Kai Lin. Delegating RAM computations with adaptive soundness and privacy. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 3–30. Springer, Heidelberg, October / November 2016.

[ACC+19] Joël Alwen, Margarita Capretto, Miguel Cueto, Chethan Kamath, Karen Klein, Ilia Markov, Guillermo Pascual-Perez, Krzysztof Pietrzak, Michael Walter, and Michelle Yeo. Keep the dirt: Tainted TreeKEM, adaptively and actively secure continuous group key agreement. Cryptology ePrint Archive, Report 2019/1489, 2019. `https://eprint.iacr.org/2019/1489`.

[ACD19] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 129–158. Springer, Heidelberg, May 2019.

[ACDT20] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 248–277. Springer, Heidelberg, August 2020.

[ACJM20] Joël Alwen, Sandro Coretti, Daniel Jost, and Marta Mularczyk. Continuous group key agreement with active security. In *TCC 2020, Theory of Cryptography Conference, Durham, NC, USA, November*, 2020.

[ACK+21] Benedikt Auerbach, Suvradip Chakraborty, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, Michael Walter, and Michelle Yeo. Inverse-sybil attacks in automated contact tracing. In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 399–421. Springer, Heidelberg, May 2021.

[ACL+14] Eric Allender, Shiteng Chen, Tiancheng Lou, Periklis A. Papakonstantinou, and Bangsheng Tang. Width-parametrized SAT: time–space tradeoffs. *Theory Comput.*, 10:297–339, 2014.

[AFGH05] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS 2005*. The Internet Society, February 2005.

[AGK+18] Joël Alwen, Peter Gazi, Chethan Kamath, Karen Klein, Georg Osang, Krzysztof Pietrzak, Leonid Reyzin, Michal Rolinek, and Michal Rybár. On the memory-hardness of data-independent password-hashing functions. In Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim, editors, *ASIACCS 18*, pages 51–65. ACM Press, April 2018.

[AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 166–184. Springer, Heidelberg, August 2013.

[AKK+19] Hamza Abusalah, Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. Reversible proofs of sequential work. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 277–291. Springer, Heidelberg, May 2019.

[AL18] Prabhanjan Ananth and Alex Lombardi. Succinct garbling schemes from functional encryption through a local simulation paradigm. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 455–472. Springer, Heidelberg, November 2018.

[AR11] Michael Alekhnovich and Alexander Razborov. Satisfiability, branch-width and tseitin tautologies. *computational complexity*, 20(4):649–678, 2011.

[AS15] Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 595–603. ACM Press, June 2015.

[AS16] Prabhanjan Vijendra Ananth and Amit Sahai. Functional encryption for turing machines. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 125–153. Springer, Heidelberg, January 2016.

[BBM+20] Richard Barnes, Benjamin Beurdouche, Jon Millican, Emad Omara, Katriel Cohn-Gordon, and Raphael Robert. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-09, Internet Engineering Task Force, March 2020. Work in Progress.

[BBR18] Karthikeyan Bhargavan, Richard Barnes, and Eric Rescorla. TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups. May 2018.

[BBS98a] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *Advances in Cryptology — EUROCRYPT'98*, pages 127–144, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[BBS98b] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 127–144. Springer, Heidelberg, May / June 1998.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer Berlin Heidelberg, 2004.

[Ben89] Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, 1989.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

[BHK13] Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Instantiating random oracles via UCEs. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 398–415. Springer, Heidelberg, August 2013.

[BHR12a] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 134–153. Springer, Heidelberg, December 2012.

[BHR12b] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012.

[BKP14] Olivier Blazy, Eike Kiltz, and Jiaxin Pan. (Hierarchical) identity-based encryption from affine message authentication. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 408–425. Springer, Heidelberg, August 2014.

[Bod88] Hans L. Bodlaender. Nc-algorithms for graphs with small treewidth. In Jan van Leeuwen, editor, *Graph-Theoretic Concepts in Computer Science, 14th International Workshop, WG '88, Amsterdam, The Netherlands, June 15-17, 1988, Proceedings*, volume 344 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 1988.

[Bod93] Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2):1–21, 1993.

[Bod98] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1):1 – 45, 1998.

[Bre74] Richard P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, April 1974.

[BRS03] John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 62–75. Springer, Heidelberg, August 2003.

[BSJ+17] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 619–650. Springer, Heidelberg, August 2017.

[BV98] Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 59–71. Springer, Heidelberg, May / June 1998.

[BV11] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Oct 2011.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013.

[CCG+18] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1802–1819. ACM Press, October 2018.

[CCL+14] Nishanth Chandran, Melissa Chase, Feng-Hao Liu, Ryo Nishimaki, and Keita Xagawa. Re-encryption, functional re-encryption, and multi-hop re-encryption: A framework for achieving obfuscation-based security and instantiations from lattices. In *Public-Key Cryptography - PKC 2014*, volume 8383, pages 95–112. Springer, March 2014.

[CCV12] Nishanth Chandran, Melissa Chase, and Vinod Vaikuntanathan. Functional re-encryption and collusion-resistant obfuscation. In Ronald Cramer, editor, *Theory of Cryptography*, pages 404–421, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[CDG01] Fan Chung, Persi Diaconis, and Ronald Graham. Combinatorics for the east model. *Advances in Applied Mathematics*, 27(1):192–206, 2001.

[CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *32nd ACM STOC*, pages 235–244. ACM Press, May 2000.

[CGI+99] Ran Canetti, Juan A. Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast security: A taxonomy and some efficient constructions. In *IEEE INFOCOM'99*, pages 708–716, New York, NY, USA, March 21–25, 1999.

[CH07] Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In *Proceedings of the 14th ACM Conference on Computer and*

Communications Security, CCS '07, pages 185–194, New York, NY, USA, 2007. ACM.

[CHK19] Cas Cremers, Britta Hale, and Konrad Kohbrok. Efficient post-compromise security beyond one group. Cryptology ePrint Archive, Report 2019/477, 2019. https://eprint.iacr.org/2019/477.

[CM01] Mary Cryan and Peter Bro Miltersen. On pseudorandom generators in NC. In *MFCS*, volume 2136 of *Lecture Notes in Computer Science*, pages 272–284. Springer, 2001.

[Coh19] Aloni Cohen. What about bob? The inadequacy of CPA security for proxy reencryption. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 287–316. Springer, Heidelberg, April 2019.

[Cor00] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 229–235. Springer, Heidelberg, August 2000.

[DKW11a] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. Key-evolution schemes resilient to space-bounded leakage. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 335–353. Springer, Heidelberg, August 2011.

[DKW11b] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable self-erasing functions. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 125–143. Springer, Heidelberg, March 2011.

[DNW05] Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 37–54. Springer, Heidelberg, August 2005.

[DS16] Léo Ducas and Damien Stehlé. Sanitization of FHE ciphertexts. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 294–310, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[DV19] F. Betül Durak and Serge Vaudenay. Bidirectional asynchronous ratcheted key agreement with linear complexity. In Nuttapong Attrapadung and Takeshi Yagi, editors, *IWSEC 19*, volume 11689 of *LNCS*, pages 343–362. Springer, Heidelberg, August 2019.

[FJP15] Georg Fuchsbauer, Zahra Jafargholi, and Krzysztof Pietrzak. A quasipolynomial reduction for generalized selective decryption on trees. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 601–620. Springer, Heidelberg, August 2015.

[FKKP18] Georg Fuchsbauer, Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. Adaptively secure proxy re-encryption. Cryptology ePrint Archive, Report 2018/426, 2018. https://eprint.iacr.org/2018/426.

[FKKP19] Georg Fuchsbauer, Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. Adaptively secure proxy re-encryption. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 317–346. Springer, Heidelberg, April 2019.

[FKPR14] Georg Fuchsbauer, Momchil Konstantinov, Krzysztof Pietrzak, and Vanishree Rao. Adaptive security of constrained PRFs. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 82–101. Springer, Heidelberg, December 2014.

[FL19] Xiong Fan and Feng-Hao Liu. Proxy re-encryption and re-signatures from lattices. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 363–382. Springer, Heidelberg, June 2019.

[FN94] Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 480–491. Springer, Heidelberg, August 1994.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[GGKT05] Rosario Gennaro, Yael Gertner, Jonathan Katz, and Luca Trevisan. Bounds on the efficiency of generic cryptographic constructions. *SIAM J. Comput.*, 35(1):217–246, 2005.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 276–288. Springer, Heidelberg, August 1984.

[GHK17] Romain Gay, Dennis Hofheinz, and Lisa Kohl. Kurosawa-desmedt meets tight security. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 133–160. Springer, Heidelberg, August 2017.

[GJ16] Anna Gál and Jing-Tang Jang. A generalization of spira's theorem and circuits with small segregators or separators. *Inf. Comput.*, 251:252–262, 2016.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th ACM STOC*, pages 365–377. ACM Press, May 1982.

[GMR01] Yael Gertner, Tal Malkin, and Omer Reingold. On the impossibility of basing trapdoor functions on trapdoor predicates. In *42nd FOCS*, pages 126–135. IEEE Computer Society Press, October 2001.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GPV07] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(133), 2007.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Adaptively secure garbling with near optimal online complexity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 535–565. Springer, Heidelberg, April / May 2018.

[GT00] Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *41st FOCS*, pages 305–313. IEEE Computer Society Press, November 2000.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011.

[HJO+16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 149–178. Springer, Heidelberg, August 2016.

[HKSS14] D. Hefetz, M. Krivelevich, M. Stojakovic, and T. Szabó. *Positional Games*. Birkhäuser Basel, 2014.

[Hof16] Dennis Hofheinz. Algebraic partitioning: Fully compact and (almost) tightly secure cryptography. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 251–281. Springer, Heidelberg, January 2016.

[Hof17] Dennis Hofheinz. Adaptive partitioning. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 489–518. Springer, Heidelberg, April / May 2017.

[HRsV07] Susan Hohenberger, Guy N. Rothblum, abhi shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. In Salil P. Vadhan, editor, *Theory of Cryptography*, pages 233–252, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 201–220. Springer, Heidelberg, May 2014.

[IN96] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9(4):199–216, September 1996.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st ACM STOC*, pages 44–61. ACM Press, May 1989.

[JKK+17a] Zahra Jafargholi, Chethan Kamath, Karen Klein, Ilan Komargodski, Krzysztof Pietrzak, and Daniel Wichs. Be adaptive, avoid overcommitting. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 133–163. Springer, Heidelberg, August 2017.

[JKK+17b] Zahra Jafargholi, Chethan Kamath, Karen Klein, Ilan Komargodski, Krzysztof Pietrzak, and Daniel Wichs. Be adaptive, avoid overcommitting. Cryptology ePrint Archive, Report 2017/515, 2017. `https://eprint.iacr.org/2017/515`.

[JMM19] Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 159–188. Springer, Heidelberg, May 2019.

[JO20] Zahra Jafargholi and Sabine Oechsner. Adaptive security of practical garbling schemes. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 741–762. Springer, Heidelberg, December 2020.

[JS14] Maurice J. Jansen and Jayalal Sarma. Balancing bounded treewidth circuits. *Theory Comput. Syst.*, 54(2):318–336, 2014.

[JS18] Joseph Jaeger and Igors Stepanovs. Optimal channel security against fine-grained state compromise: The safety of messaging. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

[JSW17] Zahra Jafargholi, Alessandra Scafuro, and Daniel Wichs. Adaptively indistinguishable garbled circuits. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 40–71. Springer, Heidelberg, November 2017.

[JW16] Zahra Jafargholi and Daniel Wichs. Adaptive security of Yao's garbled circuits. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 433–458. Springer, Heidelberg, October / November 2016.

[KKP21a] Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. On treewidth, separators and yao's garbling. TCC 2021, to appear, 2021.

[KKP21b] Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. On treewidth, separators and yao's garbling. Cryptology ePrint Archive, Report 2021/926, 2021. `https://eprint.iacr.org/2021/926`.

[KKPW21a] Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. The cost of adaptivity in security games on graphs. TCC 2021, to appear, 2021.

[KKPW21b] Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. The cost of adaptivity in security games on graphs. Cryptology ePrint Archive, Report 2021/059, 2021. `https://eprint.iacr.org/2021/059`.

[KKPW21c] Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Daniel Wichs. Limits on the adaptive security of yao's garbling. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 486–515, Virtual Event, August 2021. Springer, Heidelberg.

[KKPW21d] Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Daniel Wichs. Limits on the adaptive security of yao's garbling. Cryptology ePrint Archive, Report 2021/945, 2021. `https://eprint.iacr.org/2021/945`.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

[KPW+21] Karen Klein, Guillermo Pascual Perez, Michael Walter, Chethan Kamath, Margarita Capretto, Miguel Cueto, Ilia Markov, Michelle Yeo, Joel Alwen, and Krzysztof Pietrzak. Keep the dirt: Tainted treekem, adaptively and actively secure continuous group key agreement. In *2021 IEEE Symposium on Security and Privacy*, pages 268–284. IEEE Computer Society Press, May 2021.

[Krá01] Richard Královič. Time and space complexity of reversible pebbling. In Leszek Pacholski and Peter Ružička, editors, *SOFSEM 2001: Theory and Practice of Informatics*, pages 292–303, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[KST99] Jeong Han Kim, Daniel R. Simon, and Prasad Tetali. Limits on the efficiency of one-way permutation-based hash functions. In *40th FOCS*, pages 535–542. IEEE Computer Society Press, October 1999.

[KW19] Lucas Kowalczyk and Hoeteck Wee. Compact adaptively secure ABE for NC^1 from k-Lin. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 3–33. Springer, Heidelberg, May 2019.

[LMPP18] Daniel Lokshtanov, Ivan Mikhailin, Ramamohan Paturi, and Pavel Pudlák. Beating brute force for (quantified) satisfiability of circuits of bounded treewidth. In Artur Czumaj, editor, *29th SODA*, pages 247–261. ACM-SIAM, January 2018.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[LT79] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[LTV98] Ming Li, John Tromp, and Paul Vitányi. Reversible simulation of irreversible computation. *Physica D: Nonlinear Phenomena*, 120(1):168 – 176, 1998. Proceedings of the Fourth Workshop on Physics and Consumption.

[LV08] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In Ronald Cramer, editor, *Public Key Cryptography – PKC 2008*, volume 4939 of *Lecture Notes in Computer Science*, pages 360–379. Springer Berlin Heidelberg, 2008.

[LW14] Allison B. Lewko and Brent Waters. Why proving HIBE systems secure is difficult. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 58–76. Springer, Heidelberg, May 2014.

[Mat19] Matthew A. Weidner. Group Messaging for Secure Asynchronous Collaboration. Master's thesis, University of Cambridge, June 2019.

[mls] Message Layer Security (mls) WG. https://datatracker.ietf.org/wg/mls/about/.

[MO14] Antonio Marcedone and Claudio Orlandi. Obfuscation $\Rightarrow$ (IND-CPA security $\not\Rightarrow$ circular security). In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 77–90. Springer, Heidelberg, September 2014.

[Nor15] Jakob Nordström. *New Wine into Old Wineskins: A Survey of SomePebbling Classics with Supplemental Results*. 2015.

[Pan07] Saurabh Panjwani. Tackling adaptive corruptions in multicast encryption protocols. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 21–40. Springer, Heidelberg, February 2007.

[Pap85] Christos H. Papadimitriou. Games against nature. *Journal of Computer and System Sciences*, 31(2):288 – 301, 1985.

[Pas13] Rafael Pass. Unprovable security of perfect NIZK and non-interactive non-malleable commitments. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 334–354. Springer, Heidelberg, March 2013.

[PH70] Michael S. Paterson and Carl E. Hewitt. Record of the project mac conference on concurrent systems and parallel computation. chapter Comparative Schematology, pages 119–127. ACM, New York, NY, USA, 1970.

[PM16] Trevor Perrin and Moxie Marlinspike. The Double Ratchet Algorithm. https://signal.org/docs/specifications/doubleratchet/, 2016.

[PR18] Bertram Poettering and Paul Rösler. Towards bidirectional ratcheted key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2018.

[PWA+16] Le Trieu Phong, Lihua Wang, Yoshinori Aono, Manh Ha Nguyen, and Xavier Boyen. Proxy re-encryption schemes with key privacy from lwe. Cryptology ePrint Archive, Report 2016/327, 2016. `https://eprint.iacr.org/2016/327`.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, New York, NY, USA, 2005. ACM.

[RS86] Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

[RTV04] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 1–20. Springer, Heidelberg, February 2004.

[Rud88] Steven Rudich. *Limits on the Provable Consequences of One-way Functions*. PhD thesis, EECS Department, University of California, Berkeley, Dec 1988.

[Sav98] John E. Savage. *Models of computation - exploring the power of computing*. Addison-Wesley, 1998.

[Sco02] Mike Scott. Authenticated ID-based key exchange and remote log-in with simple token and PIN number. Cryptology ePrint Archive, Report 2002/164, 2002. `https://eprint.iacr.org/2002/164`.

[Sim98] Daniel R. Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 334–345. Springer, Heidelberg, May / June 1998.

[Spi71] P.M. Spira. On time-hardware complexity of tradeoffs for boolean functions. In *Proc. 4th Hawaii Symp. System Sciences*, page 525–527. North Hollywood and Western Periodicals, 1971.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

[WGL98] Chung Kei Wong, Mohamed G. Gouda, and Simon S. Lam. Secure group communications using key graphs. In *Proceedings of ACM SIGCOMM*, pages 68–79, Vancouver, BC, Canada, August 31 – September 4, 1998.

[WGL00] Chung Kei Wong, Mohamed G. Gouda, and Simon S. Lam. Secure group communications using key graphs. *IEEE/ACM Trans. Netw.*, 8(1):16–30, 2000.

[WHA98] D. M. Wallner, E. J. Harder, and R. C. Agee. Key management for multicast: Issues and architectures. Internet Draft, September 1998. `http://www.ietf.org/ID.html`.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

APPENDIX A

Appendix

A.1 Optimize Lemma 23 and Corollary 4 from Chapter 5

In the proof of Lemma 23 we considered a variant GSD′ of GSD, where the challenger aborts once event E is triggered. Here we optimize the result by prooving that GSD′-security reduces to IND-CPA security of the underlying encryption scheme at a loss in security which is *independent* of the degree of the key graph.

Lemma 54. *Let the event E be defined as in the proof of Theorem 17 and* A *an adversary against* GSD′ *with advantage ϵ, where* GSD′ *is defined similar to GSD but the challenger aborts once event E is triggered. Then, in the random oracle model, there exists an IND-CPA adversary* A′ *(that has essentially the same running time) against the underlying encryption scheme with advantage $> \frac{\epsilon}{N^2}$, where N is the number of nodes.*

Proof. We define an adversary A′ against IND-CPA security of Π as follows (see Figure A.1): On receipt of a challenge public key pk^*, A′ samples two independent uniformly random seeds $s, s' \in \mathcal{S}$ and sends them to the IND-CPA challenger. In response it receives a challenge ciphertext c^* which encrypts either s in case $b^* = 0$, or s' if $b^* = 1$. To guess the secret bit b^*, the algorithm A′ now attempts to simulate a hybrid game to A and embed the challenge c^* in such a way that, for some i, the execution of the game will look just like G'_{i-1} if $b^* = 0$ and like G'_i if $b^* = 1$ (where G'_i is a variant of G_i from the proof of Theorem 17 where the game aborts in case E happens). To this aim, A′ first samples $v^* \leftarrow [1, N]$ uniformly at random (i.e., it guesses the challenge node). Then it samples independent seeds $s_u \in S$ uniformly at random for all nodes $u \in [1, N] \setminus \{v^*\}$ and sets $s_{v^*} := s$. Instead of sampling a fresh seed s', A′ just uses the seed s' which it used in the IND-CPA game. Hence, both s_{v^*} and s' are associated with node v^*. Next, A′ chooses $u^* \leftarrow [1, N]$ uniformly at random and for all $u \in [1, N] \setminus \{u^*\}$ queries the random oracle H on s_u to receive $r_u := H(s_u)$. Just the same as in the GSD game, A' then computes $(\mathrm{pk}_u, \mathrm{sk}_u) \leftarrow \mathsf{Gen}(r_u)$. To embed the challenge, it sets $\mathrm{pk}_{u^*} := \mathrm{pk}^*$.

A′ now answers all encryption queries just as in the game GSD_0 except for the edges incident on v^*: All queries $(\mathtt{encrypt}, u, v^*)$ are answered with encryptions $\mathsf{Enc}_{\mathrm{pk}_u}(s')$, until A queries $(\mathtt{encrypt}, u^*, v^*)$, which is answered with c^*; all further queries are answered with encryptions of s. All corrupt queries are answered just the same as in the GSD game and the challenge

query v is answered with s_v. For all queries A makes to the random oracle, A′ just queries its own oracle H and returns the output.

In this way A′ aims to simulate some game G'_i and if the guess v^* was correct this simulation is perfect, except for the case that A queries the random oracle H on s_{u^*}: To answer this query consistently, A′ would have to output a seed r_{u^*} satisfying $\mathsf{Gen}(r_{u^*}) = (\mathtt{pk}^*, \mathtt{sk}^*)$. This however implies that either 1) the node u^* was challenged or corrupted, hence, u^* can not be a parent of the challenge node so the simulation failed due to an incorrect guess anyway; or 2) A triggered event E, in which case G'_i aborts and thus A′ can simply abort and output 1, which faithfully preserves the simulation. Accordingly, as soon as A′ sees that its guesses u^* and v^* are incorrect, i.e., v^* is not the challenge node or u^* is corrupted or reachable from a corrupted node, it aborts the game and outputs $b' = 1$.

The following events will now determine the output behaviour of A′:

- Event U: A queried $(\mathtt{encrypt}, u^*, v)$, where v denotes the challenge node.
- Event V: The challenge node v coincides with v^*.

After running the game with A, the reduction A′ evaluates whether these events happened and chooses its final output b' as follows: If U and V are true, this means that A′ correctly guessed the challenge node v, managed to embed the IND-CPA challenge key $\mathtt{pk}^*$ at u^* and some game G'_i was executed. In this case A′ outputs the bit b it received from A. Note that if $(\mathtt{encrypt}, u^*, v^*)$ happens to be the first encryption query incident on the challenge node $v = v^*$ and $b^* = 0$, then the game simulated to A looks exactly the same as GSD'_0. If $(\mathtt{encrypt}, u^*, v^*)$ happens to be the last encryption query incident on the challenge node $v = v^*$ and $b^* = 1$, then the game simulated to A looks exactly the same as GSD'_1. If either U or V fails, as mentioned above, A' outputs $b' = 1$.

If A either doesn't make a challenge query or the challenge node v is a source node, then the advantage $\tilde{\epsilon}$ of A in winning the GSD game is 0 (information-theoretically; note by definition the challenge node must be a sink). Hence, we assume that v is not a source node. Furthermore, we assume that v constitutes a valid GSD challenge, since otherwise, again, the advantage of A is 0. If any of these assumptions fail, A′ outputs $b' = 1$.

For the advantage of A′ in breaking the IND-CPA security of the encryption scheme we have

$$\begin{aligned}\Pr[\mathsf{A}' \to 0 \mid b^* = 0] - \Pr[\mathsf{A}' \to 0 \mid b^* = 1] \\ = \Pr[(\mathsf{A}' \to 0) \wedge U \wedge V \mid b^* = 0] - \Pr[(\mathsf{A}' \to 0) \wedge U \wedge V \mid b^* = 1].\end{aligned}$$

For technical reasons, we slightly modify A′ as follows: After the interaction with A, the reduction A′ one by one adds further edges incident on v, where it starts with edges outgoing from nodes which are neither corrupted nor reachable from a corrupted node, if such nodes exist. This neither changes the output behaviour of A′ nor validity of the challenge node, since it happens after the execution of the GSD′ game. We consider $N - 1$ disjoint events U_i denoting the event that u^* is the source of the ith encryption edge incident on v (with respect to the chronological order of the queries). Let δ_{in} be an upper bound on the indegree of the (final) key graph G. We denote the event that A queries exactly j encryptions incident on the challenge node by Δ_j. Hence, if V, U_i and Δ_j for some $j \geq i$ happen, then depending on the challenge bit b^*, the game simulated to A is distributed exactly the same as G'_{i-1} or G'_i. Since V and $\mathsf{A}' \to 0$ implies U and we are guaranteed that one of the U_i and one of the Δ_j

Algorithm A.1: Security reduction R for proof in the random oracle model. C is the IND-CPA challenger, the oracles are defined in A.2.

$\mathsf{R}^{\mathsf{C},\mathsf{A},H}_{\Pi=(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})}(\mathtt{pk}^*)$

$\hat{U}, V, E, \mathsf{FAIL} \leftarrow \mathsf{FALSE}$
$\mathcal{E}, \mathcal{C} \leftarrow \emptyset$, $G \leftarrow ([1,N], \mathcal{E})$ `// initialize key graph, set of corrupt nodes`
$s^* \leftarrow \perp$
$v^* \leftarrow [1,N]$ `// guess challenge node`
$s, s' \leftarrow \mathcal{S}$ `// choose challenge seeds`
$(\mathtt{pk}^*, \mathtt{c}^*) \leftarrow^{\$} \mathsf{C}(s, s')$ `// receive IND-CPA challenge`
for $u \in [1,N] \setminus \{v^*\}$ **do** `// sample remaining seeds`
  $s_u \leftarrow \mathcal{S}$
$s_{v^*} \leftarrow s$
$u^* \leftarrow [1,N]$ `// guess parent of challenge node`
for $u \in [1,N] \setminus \{u^*\}$ **do** `// sample keys`
  $r_u \leftarrow H(s_u)$
  $(\mathtt{pk}_u, \mathtt{sk}_u) \leftarrow^{\$} \mathsf{Gen}(r_u)$
$\mathtt{pk}_{u^*} \leftarrow \mathtt{pk}^*$ `// embed challenge key`
$b \leftarrow \mathsf{A}^{\mathcal{O}\text{-enc},\mathcal{O}\text{-corr},\mathcal{O}\text{-chall},\mathcal{O}\text{-}H}$
if *(no challenge query was issued) or (v reachable in G from a node in $\mathcal{C}$) or (v not a sink node in G) or (v a source node in G)* **then**
  $\mathsf{FAIL} \leftarrow \mathsf{TRUE}$
if $\neg V$ *or* $\neg \hat{U}$ *or* E *or* FAIL **then**
  return 1
else
  return b

happens, we have for all i, j:

$$\Pr[(\mathsf{A}' \to 0) \wedge U \wedge V \mid b^* = 0] = \sum_{j=1}^{\delta_{in}} \sum_{i=1}^{N-1} \Pr[\mathsf{A}' \to 0 \mid \Delta_j \wedge V \wedge U_i \wedge (b^* = 0)] \cdot \Pr[\Delta_j \mid V \wedge U_i \wedge (b^* = 0)] \cdot \Pr[V \wedge U_i \mid b^* = 0] \tag{A.1}$$

and similarly for the case $b^* = 1$. Let U^* denote the subevent of U that u^* is neither corrupted nor reachable from a corrupted node. Then for all $i \in [2, N-1]$, (since E doesn't happen) the game simulated to A is identically distributed in the two cases $U^* \wedge U_i \wedge (b^* = 0)$ and $U^* \wedge U_{i-1} \wedge (b^* = 1)$. This is true no matter if the ith encryption was queried by A or not. Thus, we have

$$\Pr[\mathsf{A}' \to 0 \mid \Delta_j \wedge V \wedge U^* \wedge U_i \wedge (b^* = 0)] = \Pr[\mathsf{A}' \to 0 \mid \Delta_j \wedge V \wedge U^* \wedge U_{i-1} \wedge (b^* = 1)].$$

On the other hand, if U^* is false and u^* is corrupted or reachable by a corrupted node, then A' outputs 1, independently of any other events to happen. For the probability that U^* happens, note that if A makes at least i encryption queries incident on the challenge node v, then U^* must be true in both cases $V \wedge U_i \wedge (b^* = 0)$ and $V \wedge U_{i-1} \wedge (b^* = 1)$. Thus, we obtain for

Algorithm A.2: Oracles for the security reduction in the random oracle model.

$\mathcal{O}$-enc(u_1, u_2)
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(u_1, u_2)\}$
if $u_2 = v^*$ **then**
  if $u_1 = u^*$ **then**
    $\hat{U} \leftarrow$ TRUE
    return c^*
  else if $\hat{U}$ **then**
    return $\mathsf{Enc}_{\mathrm{pk}_{u_1}}(s)$
  else
    return $\mathsf{Enc}_{\mathrm{pk}_{u_1}}(s')$
return $\mathsf{Enc}_{\mathrm{pk}_{u_1}}(s_{u_2})$

$\mathcal{O}$-corr(u)
$\mathcal{C} \leftarrow \mathcal{C} \cup \{u\}$
return s_u

$\mathcal{O}$-chall(v) `// one time use only`
if $v = v^*$ **then**
  $V \leftarrow$ TRUE
$\mathcal{C} \leftarrow \mathcal{C} \cup \{v\}$
return s_v

$\mathcal{O}$-$H(\bar{s})$
if $\bar{s} = s_{u^*}$ **then**
  FAIL $\leftarrow$ TRUE
  return $\perp$
if *($\bar{s} = s_u$ for some $u \in [1, N] \setminus \{v\}$) or (not V and $\bar{s} = s'$)* **then**
  if *u (resp. v) is not reachable in G from any of the $u' \in \mathcal{C}$* **then**
    $E \leftarrow$ TRUE
    abort A
return $H(\bar{s})$

all $j \in [1, \delta_{in}]$, $i \in [2, N-1]$ such that $j \geq i$:

$$\begin{aligned}\Pr[\mathsf{A}' \to 0 \,|\Delta_j \wedge V \wedge U_i \wedge (b^* = 0)] \\ &= \Pr[\mathsf{A}' \to 0 \mid \Delta_j \wedge V \wedge U^* \wedge U_i \wedge (b^* = 0)] \cdot \Pr[U^* \mid \Delta_j \wedge V \wedge U_i \wedge (b^* = 0)] \\ &= \Pr[\mathsf{A}' \to 0 \mid \Delta_j \wedge V \wedge U^* \wedge U_{i-1} \wedge (b^* = 1)] \cdot \Pr[U^* \mid \Delta_j \wedge V \wedge U_{i-1} \wedge (b^* = 1)] \\ &= \Pr[\mathsf{A}' \to 0 \mid \Delta_j \wedge V \wedge U_{i-1} \wedge (b^* = 1)].\end{aligned} \tag{A.2}$$

If A queries less than i encryption queries incident on v, then U_i implies that A′ outputs 1, independently of b^*. Thus, for all $j \in [1, \delta_{in}]$, $i \in [2, N-1]$ such that $j < i$:

$$\Pr[\mathsf{A}' \to 0 \mid \Delta_j \wedge V \wedge U_i \wedge (b^* = b)] = 0 \tag{A.3}$$

for either $b \in \{0,1\}$. By a similar argument as above, we have for all $j > i$:

$$\Pr[\Delta_j \mid V \wedge U_i \wedge (b^* = 0)] = \Pr[\Delta_j \mid V \wedge U_{i-1} \wedge (b^* = 1)] \tag{A.4}$$

In fact, (A.4) also holds in case $j = i$, but this is more subtle. Here, it is crucial that A' starts with uncorrupted nodes when choosing additional edges after the interaction with A: Clearly, if there are only i nodes (including v) which are neither corrupted nor reachable from a corrupted node, then Δ_i cannot happen. On the other hand, if there are more than i such nodes, then U_i implies that u^* is such a node and the claim follows by noting that after the $i-1$th query the case $V \wedge U_{i-1} \wedge (b^* = 1)$ is indistinguishable from the case where the edge (u^*, v^*) has not yet been made.

Since U_i is equally distributed for all $i \in [1, N-1]$ and independent of b^*, it also holds

$$\begin{aligned}
\Pr[V \wedge U_i \mid b^* = 0] &= \Pr[V \mid U_i \wedge (b^* = 0)] \Pr[U_i \mid b^* = 0] \\
&= \Pr[V \mid U_i \wedge (b^* = 0)] \Pr[U_i] \\
&= \Pr[V \mid U_{i-1} \wedge (b^* = 1)] \Pr[U_{i-1}] \\
&= \Pr[V \mid U_{i-1} \wedge (b^* = 1)] \Pr[U_{i-1} \mid b^* = 1] \\
&= \Pr[V \wedge U_{i-1} \mid b^* = 1].
\end{aligned} \tag{A.5}$$

This implies

$$\begin{aligned}
&\sum_{j=1}^{\delta_{in}} \sum_{i=1}^{N-1} \Pr[\mathsf{A}' \to 0 \mid \Delta_j \wedge V \wedge U_i \wedge (b^* = 0)] \cdot \Pr[\Delta_j \mid V \wedge U_i \wedge (b^* = 0)] \cdot \Pr[V \wedge U_i \mid b^* = 0] \\
&= \sum_{j=1}^{\delta_{in}} \Big(\Pr[\mathsf{A}' \to 0 \mid \Delta_j \wedge V \wedge U_1 \wedge (b^* = 0)] \cdot \Pr[\Delta_j \mid V \wedge U_1 \wedge (b^* = 0)] \cdot \Pr[V \wedge U_1 \mid b^* = 0] \\
&\qquad + \sum_{i=2}^{j} \Pr[\mathsf{A}' \to 0 \mid \Delta_j \wedge V \wedge U_{i-1} \wedge (b^* = 1)] \cdot \Pr[\Delta_j \mid V \wedge U_{i-1} \wedge (b^* = 1)] \\
&\qquad\qquad \cdot \Pr[V \wedge U_{i-1} \mid b^* = 1] \Big)
\end{aligned} \tag{A.6}$$

Thus, when computing the difference of the probabilities of $(\mathsf{A}' \to 0) \wedge V \wedge U$ conditioned on $b^* = 0$ and $b^* = 1$, respectively, most terms cancel, except for the terms for the two extreme cases $U_1 \wedge (b^* = 0)$ and $U_j \wedge (b^* = 1)$.

$$\begin{aligned}
&\Pr[(\mathsf{A}' \to 0) \wedge U \wedge V \mid b^* = 0] - \Pr[(\mathsf{A}' \to 0) \wedge U \wedge V \mid b^* = 1] \\
&= \sum_{j=1}^{\delta_{in}} (\Pr[(\mathsf{A}' \to 0) \wedge \Delta_j \wedge V \wedge U_1 \mid b^* = 0] - \Pr[(\mathsf{A}' \to 0) \wedge \Delta_j \wedge V \wedge U_j \mid b^* = 1]) \\
&= \Pr[(\mathsf{A}' \to 0) \wedge V \wedge U_1 \mid b^* = 0] - \Pr[(\mathsf{A}' \to 0) \wedge V \wedge U_{\mathsf{last}} \mid b^* = 1],
\end{aligned} \tag{A.7}$$

where U_{last} denotes the event that u^* is the source of the last encryption query incident on v which A makes. If V and U_1 happen and $b^* = 0$, then the game simulated to A is exactly the same as the real game GSD'_0; similarly, if V and U_{last} happen and $b^* = 1$, then the game simulated to A is exactly the same as the random game GSD'_1. In other words,

$$\Pr[\mathsf{A}' \to 0 \mid V \wedge U_1 \wedge (b^* = 0)] = \Pr[\mathsf{A}(\mathsf{GSD}'_0) \to 0]$$

and

$$\Pr[\mathsf{A}' \to 0 \mid V \wedge U_{\mathsf{last}} \wedge (b^* = 1)] = \Pr[\mathsf{A}(\mathsf{GSD}'_1) \to 0]. \tag{A.8}$$

Now, as long as A does not issue the challenge query, the games simulated to A are identically distributed in both cases[1] $V \wedge U_1 \wedge (b^* = 0)$ and $V \wedge U_{\mathsf{last}} \wedge (b^* = 1)$: All edges incident on v^* encrypt the same seed (s in the first case and s' in the latter), and both seeds associated with v^* were neither queried to H nor reveiled by a corruption or challenge query. Hence, v^* looks just the same as any other potential challenge node and we have

$$\Pr[V \mid U_1 \wedge (b^* = 0)] = \Pr[V \mid U_{\mathsf{last}} \wedge (b^* = 1)] \geq \frac{1}{N}. \tag{A.9}$$

Since we're guaranteed that the challenge node will not be a source node, we also have that

$$\Pr[U_1 \mid b^* = 0] = \Pr[U_{\mathsf{last}} \mid b^* = 1] \geq \frac{1}{N}. \tag{A.10}$$

Combining equations (A.1)-(A.10), we finally arrive at

$$\begin{aligned}
&\Pr[(\mathsf{A}' \to 0) \wedge U \wedge V \mid b^* = 0] - \Pr[(\mathsf{A}' \to 0) \wedge U \wedge V \mid b^* = 1] \\
&= \Pr[\mathsf{A}' \to 0 \mid V \wedge U_1 \wedge (b^* = 0)] \cdot \Pr[V \wedge U_1 \mid b^* = 0] \\
&\quad - \Pr[\mathsf{A}' \to 0 \mid V \wedge U_{\mathsf{last}} \wedge (b^* = 1)] \cdot \Pr[V \wedge U_{\mathsf{last}} \mid b^* = 1] \\
&= \Big(\Pr[\mathsf{A}' \to 0 \mid V \wedge U_1 \wedge (b^* = 0)] - \Pr[\mathsf{A}' \to 0 \mid V \wedge U_{\mathsf{last}} \wedge (b^* = 1)] \Big) \cdot \Pr[V \wedge U_1 \mid b^* = 0] \\
&\geq \frac{1}{N^2} \cdot \Big(\Pr[\mathsf{A}(\mathsf{GSD}'_0) \to 0] - \Pr[\mathsf{A}(\mathsf{GSD}'_1) \to 0] \Big).
\end{aligned}$$

This proves the claim. □

Corollary 21. *Let the event E be defined as in the proof of Theorem 17, and* A *an arbitrary GSD adversary which triggers E with probability ϵ. Then there exists an IND-CPA adversary* A′ *(that has essentially the same running time) with advantage*

$$\Pr[\mathsf{A}' \to 0 \mid b^* = 0] - \Pr[\mathsf{A}' \to 0 \mid b^* = 1] \geq \frac{\epsilon}{N^2} - \frac{m}{N2^\lambda},$$

where N is the number of nodes, m the number of oracle queries to H, and λ the seed length.

Proof. We first apply Lemma 24 to construct an adversary against GSD″ with advantage $\frac{\epsilon}{N} - \frac{m}{2^\lambda}$. Then we apply a very similar proof as for Lemma 54 to construct an IND-CPA adversary: This proof is exactly the same with the only difference being that GSD″ is less adaptive, i.e. A commits to the challenge node at the beginning of the game. This means that A′ does not have to guess v^* and the event V is always true. Thus, we only lose an additional factor $1/N$ due to guessing u^*. Finally, we note that, since A′ is sampling the keys itself for all nodes except u^*, the increase in the number of nodes from N to $N + 1$ due to Lemma 24 can be ignored, since A′ does not actually need to create the additional node, since it knows the seed s_{v^*}. □

[1] Note that not all encryption queries incident on v might have been issued at this point, but this does not affect the analysis.