The Cost of Adaptivity in Security Games on Graphs

Chethan Kamath*[1], Karen Klein†[2], Krzysztof Pietrzak†[2], and Michael Walter†[2]

[1]ckamath@protonmail.com
[2]IST Austria, {kklein,pietrzak,mwalter}@ist.ac.at

July 8, 2021

Abstract

The security of cryptographic primitives and protocols against adversaries that are allowed to make adaptive choices (e.g., which parties to corrupt or which queries to make) is notoriously difficult to establish. A broad theoretical framework was introduced by Jafargholi et al. [Crypto'17] for this purpose. In this paper we initiate the study of lower bounds on loss in adaptive security for certain cryptographic protocols considered in the framework. We prove lower bounds that almost match the upper bounds (proven using the framework) for proxy re-encryption, prefix-constrained PRFs and generalized selective decryption, a security game that captures the security of certain group messaging and broadcast encryption schemes. Those primitives have in common that their security game involves an underlying graph that can be adaptively built by the adversary.

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 can 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.

*Most of the work was done while the author was at Northeastern University, supported by the IARPA grant IARPA/2019-19-020700009, and Charles University, funded by project PRIMUS/17/SCI/9.

†Funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (682815 - TOCNeT).

Contents

1 Introduction

Consider the following game played between a challenger C and an adversary A using a symmetric-key encryption (SKE) 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 $U_1, \ldots, U_N$ 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 $\mathsf{Enc}(\mathtt{k}_i, \mathtt{k}_j)$, or
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 U_{i^*} and is given either the real key $\mathtt{k}_{i^*}$ or an independent, random key 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 can simply decrypt the ciphertexts along the path from any corrupted node to the challenge to learn $\mathtt{k}_{i^*}$.

The above game is called *generalised selective decryption* (GSD) and it captures the security of protocols for multicast encryption [Pan07] and continuous group key agreements (CGKA) [ACDT20, ACC+19]. We will use GSD in this introduction as the running example to convey our ideas. The main question regarding GSD 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 SKE.[1] For this we need to prove a computational soundness (i.e., security) theorem of the form: if the SKE is ϵ-secure then the GSD

[1]In case the key-graph contains cycles, one must additionally assume that the SKE 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 where the acyclicity is enforced by the protocol rules.

game is ϵ'-secure for some ϵ' that depends on ϵ. Ideally, the loss of security[2] should be kept to a polynomial, i.e., $\epsilon' = \epsilon/\mathbf{poly}(N)$. Otherwise, this requires to either set the security parameter of the underlying SKE very large if one wants to maintain provable security guarantees, which will lead to inefficiency. Or the provided security is only heuristic, leaving the possibility of an attack against GSD which does not break the underlying SKE.

The simpler task of proving 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. If the graph is known ahead of time, it is easy to construct a series of $O(N^2)$ hybrids, each of which can be shown indistinguishable under the security of the SKE (see, e.g., [JKK+17]). 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 in [Pan07] and remains notoriously hard. In particular, non-trivial results are only known in settings, where the adversary is restricted to query (subgraphs of) specific key-graphs (which needs to be enforced by the higher level protocol). The state of the art is represented by the general Piecewise-Guessing framework [JKK+17, KW19].

1.1 Our Results

The Piecewise-Guessing Framework has been successfully used to give improved security guarantees against adaptive attacks for various applications [KW19, FKKP19, ACC+19, ACDT20, KNYY20], but there still are significant gaps to knows attacks. In this paper we approach this question from the other "lower bounds" direction, and for several applications show that this will not be possible, at least not when using existing techniques. In particular, we show that there do not exist efficient non-rewinding black-box reductions – henceforth called "straight-line" reductions for brevity – that prove security of

- certain forms of restricted GSD (including its public key variant) based on the IND-CPA security of the underlying SKE (see Section 6),
- popular protocols for CGKA based on the IND-CPA security of the underlying public-key encryption (PKE) (see Section 7),
- the GGM construction for prefix-constrained PRFs based on the pseudorandomness of the underlying PRG (see Section 8)
- proxy re-encryption[3] (PRE) schemes [BBS98] based on the IND-CPA security of the PKE and N-weak key privacy (see Section 9)

[2]Consider a reduction R from a problem P to another problem Q. Suppose that R (ϵ', T')-breaks P when given access to an (ϵ, T)-adversary A that breaks Q. The *loss in security* Λ incurred by R is defined as the ratio $(\epsilon T')/(\epsilon' T)$.

[3]A proxy re-encryption scheme is a public-key encryption scheme that allows the holder of a key $\mathtt{pk}$ to derive a re-encryption key for any other 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. The formal definition is given in Section 9.

Application	Underlying Graph	Lower Bound	Reduction	Upper Bound
GSD	Path P_N	$N^{\Omega(\log(N))}$	Oblivious	$N^{O(\log(N))}$ [FJP15]
	Rooted Binary In-Tree B_n	$N^{\Omega(\log(N))}$	Oblivious	$N^{O(\log(N))}$ [Pan07]
	Tree[4]	$N^{\Omega(\log(N))}$	Straight-line	$N^{O(\log(N))}$ [FJP15]
	Arbitrary DAG	$2^{\Omega(\sqrt{N})}$	Oblivious	$N^{O(N/\log(N))}$ [JKK+17]
PRE	Path P_N	$N^{\Omega(\log(N))}$	Oblivious	$N^{O(\log(N))}$ [FKKP19]
	Binary Tree B_n	$N^{\Omega(\log(N))}$	Oblivious	$N^{O(\log(N))}$ [FKKP19]
	Arbitrary DAG	$2^{\Omega(N)}$	Arbitrary	$N^{O(N/\log(N))}$ [FKKP19]
GGM CPRF	Tree	$n^{\Omega(\log(n))}$	Straight-line	$n^{O(\log(n))}$ [FKPR14]
TreeKEM	Regular Tree	$M^{\Omega(\log(\log(M)))}$	Straight-line	$Q^{O(\log(M))}$ [ACC+19]

Table 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.

with only polynomial loss in advantage. For PRE we can even rule out general (i.e., rewinding) black-box reductions (see Discussion of Corollary 14). 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.

Theorem 1 (Informally Stated, Corollary 5)**.** *Any straight-line reduction proving security of unrestricted adaptive* GSD *based on the IND-CPA security of the underlying SKE 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 [JKK+17], 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 [FKKP19]), as stated informally below.

Theorem 2 (Informally Stated, Corollaries 2 to 4)**.** *Any oblivious reduction proving security of adaptive GSD restricted to* paths or binary trees *based on the IND-CPA security of the underlying SKE 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

[4]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 reduction is naturally more restricted. A summary of the results can be found in Table 1.

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) as their security model is not quite captured by the Builder-Pebbler Game. The high level reason for this is that the graphs (e.g., circuit to be garbled or the access structure) in these applications is fixed ahead of the 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 work, [KKPW] showed lower bounds for adaptive security of Yao's garbling (see Section 1.2.2 for a comparison). We defer the discussion on the Builder-Pebbler Game to the next section (Section 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 [JKK+17]. 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*.[5] 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 21; the corresponding definition for PREs is given in Definition 34.

[5]Technically, we do not require *minimal* vertex cover, but a weaker notion which we call "non-trivial" vertex cover (see Definition 2).

Why oblivious reductions? We note that oblivious 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 analysed in [JKK$^+$17] (and its follow-up works), partitioning-based reductions, which have been successfully employed in a plethora of works [Cor00], also roughly behave in an oblivious manner.[6] Moreover, oblivious 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 10.

1.2 Related Work

1.2.1 Adaptive Security

The security of multi-party computation in the context of adaptive corruption has been well studied. It is known that a protocol that is proven secure against static (i.e., non-adaptive) adversaries may turn out insecure once the adversary is allowed adaptive corruption [CFGN96]. On the other hand, in the (programmable) random oracle model it *is* possible to compile a selective protocol into an adaptively-secure one through non-committing encryption [Nie02].

The notion of generalised selective decryption (GSD) was introduced by Panjwani [Pan07] to study adaptive corruption in restricted settings. His motivation was to better understand the problem of selective decommitment [DNRS99] (which is also known as selective opening in some works [BHY09]) and the closely-related problem of selective decryption. The problem was further studied by Fuchsbauer et al. [FJP15] who gave a quasi-polynomial reduction when the GSD game is restricted to trees.

In parallel, the study of adaptive security in the setting of circuit garbling was undertaken in the works of Bellare et al. [BHR12], Hemenway et al. [HJO$^+$16] and Jafargholi and Wichs [JW16]. The latter two works are especially relevant since they established a relationship between adaptive security and graph pebbling. It is also worth noting that the study of adaptive security of garbled RAM was carried out in [GS18, GOS18].

The above two series of works culminated in the Piecewise-Guessing Framework of Jafargholi et al. [JKK$^+$17] who managed to abstract out the ideas therein and give even more fine-grained reductions. In addition to capturing the results from [JW16, FJP15, FKPR14], they applied the framework to obtain new results for adaptive secret sharing. The framework was further applied to argue adaptive security for attribute-based encryption schemes [KW19], proxy re-encryption schemes

[6]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.

[FKKP19], continuous group key-agreement [ACC$^+$19, ACDT20] and non-interactive zero-knowledge [KNYY20].

1.2.2 Limitations of Reductions

The study of limitations of reductions (see Footnote 10) was initiated in the seminal work of Impagliazzo and Rudich [IR89]. They used *oracle separations* to rule out fully black-box reduction of key agreement to symmetric-key primitives. This approach turned out quite useful and has been further exploited to rule out fully black-box reduction of a variety of cryptographic primitives from one another (e.g., [Rud88, Sim98]). A fine-grained study of the notion of reductions and separations was later carried out by Reingold et al. [RTV04].

In addition to ruling out reductions, the more fine-grained question of efficiency of reduction of one primitive to another has also been studied [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.

Comparison with [KKPW]. Building on the high level ideas in this paper, [KKPW] showed lower bounds on the adaptive security of Yao's garbling scheme. As pointed out in the introduction, the graph (i.e., the circuit) in Yao's garbling scheme is fixed ahead of time and the adaptivity comes from the choice of (garbling) input. (The difficulty of the reduction comes from having to guess the bits running over a subset of wires during evaluation of the circuit.) Therefore they had to rely on a different combinatorial abstraction from Builder-Pebbler Game (viz., a black-gray pebble game on the circuit) to establish their lower bound. However, since the security game for Yao's garbling consists of just two rounds, [KKPW] did not encounter some of the difficulties (to do with the multiple rounds of interaction) we do and therefore were able to rule out arbitrary black-box reductions. While both [KKPW] and this work model choices made by a reduction by putting pebbles on a graph structure, the analogy basically ends there. None of the main ideas from [KKPW] seem applicable in this setting and vice versa.

1.2.3 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, DKW11, 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 [JKK+17]. 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.

2 Technical Overview

On a high level, our approach can be divided into two steps. In the first step (Section 2.2), which is purely combinatorial, we analyse 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 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 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.

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 6, any reduction attempting to take advantage of our adversaries must send it's 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. It's 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

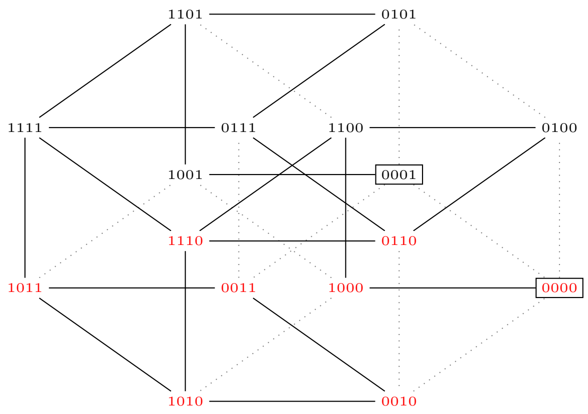


Figure 1: Configuration graph for paths of length four, $C_4 = ([5], \{(1,2),(2,3),(3,4),(4,5)\})$. It is a subgraph of the Boolean hypercube of dimension four (the missing edges are dotted). The labels of the vertices encode the pebbling status of the corresponding edge and therefore represents a pebbling 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 consists of the set of (red) vertices that lie on the 'bottom' half of the graph: $\{0000, 0010, 1010, 1011, 0011, 1000, 0110, 1110\}$. The set of edges from the top half to the bottom half form cut set: $\{(1111, 1110), (1111, 1011), (1100, 1110), (1100, 1000), (0111, 0110), (0100, 0110)\}$. 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 2.2 and 5).

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 reductions 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 1. A more formal definition is given later in Definition 5 (Section 3). A path from $\mathcal{P}_{\text{start}}$ to $\mathcal{P}_{\text{target}}$ corresponds to a pebbling sequence 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, JKK+17]. 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 "bottle-

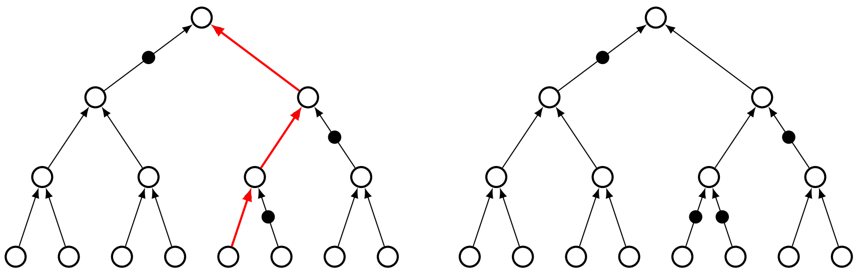


Figure 2: The pebbling configurations used to argue lower bounds for edge-pebbling of a perfect binary tree B_3 of depth 3. In the left configuration there exists a path from a leaf (100) to the root (ε) that is *not* covered by a pebble highlighted by the thicker (red) path. In the right configuration *all* the paths in B_3 are covered by pebbles. The cut is defined at such two configurations.

neck" 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 cuts 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 beginning of the sequence *none* of the $N/2$ paths from the root to the leaves 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, decrease 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 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 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 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 2.3).

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 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.[7]

Defining winning configurations via cuts of 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

[7]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 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.

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 is that in all our applications, if the reduction 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 Theorems 3 to 5 are (almost) tight since a random Pebbler yields (almost) matching lower bounds.

Theorem 3 (Informally Stated, Theorems 6 and 8)**.** *Any oblivious*[8] *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 .*

Theorem 4 (Informally Stated, Theorem 9)**.** *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.*

Theorem 5 (Informally Stated, Theorem 11)**.** *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 1 (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 4.1).[9] The reason we restrict ourselves to such Builders is mostly for our

[8]The notion of obliviousness for Pebblers is naturally derived from the one for reductions, see discussion above and Definition 10.

[9]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].

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.

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 the reduction[10] of a primitive Q (e.g., PKE) to 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 any efficient black-box reduction R that breaks P when given black-box access to an adversary that breaks Q^P (i.e., from P to 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.[11] To this end, we establish a tight *coupling* between the security game for Q^P and the Builder-Pebbler Game (e.g., Lemma 7). 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

[10]The usage of the word 'reduction' here and in Section 1.2.2 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. In the rest of the paper, 'reduction' is used to refer to a security reduction as in (ii).

[11]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)* P *of* P *and for every (not necessarily efficient) adversary* A_Q *that breaks* Q^{P} *such that the loss in security is at most* Λ.

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 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 Theorem 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 $\mathsf{Enc}(\mathtt{k}_u, \mathtt{k}_v)$ 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 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 2.1: i.e., it outputs 0 if $\mathcal{P}$ is on one side of the cut and 1 otherwise (see Figure 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:

- Enc is a random expanding function that implements encryption and
- Dec is the decryption function defined to be "consistent" with Enc.[12]

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 one side of the cut to the other. 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 4). If particular, if Λ is significantly less than quasi-polynomial in $N = 2^n$, it would imply the existence of a Pebbler that is successful

[12] 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.

with a probability greater than inverse quasi-polynomial, a contradiction to Theorem 3 (formally, Corollary 3). Since Theorem 3 only holds for oblivious Pebblers, the bound on Λ only holds for oblivious GSD reductions.

3 Preliminaries

We use the notation $[N] = \{1, \ldots, N\}$ and $[N]_0 = \{0\} \cup [N]$. For a string $x = x_0, \ldots, x_{n-1} \in \{0,1\}^n$, for $0 \le a \le b < n$, we use $x[a,b]$ to denote the substring $x_a, \ldots, x_b$.

3.1 Graph Theory

Let $N \in \mathbb{N}$ and $G = (\mathcal{V}, \mathcal{E})$ define a directed acyclic graph (DAG) with vertex set $\mathcal{V} = [N]$, edge set $\mathcal{E} \subset [N] \times [N]$, and a set of sinks $\mathcal{T}$. For a subset $\mathcal{S} \subseteq [N]$ of nodes, let $\mathrm{in}(\mathcal{S})$ denote the set of ingoing edges and $\mathrm{parents}(\mathcal{S})$ denote the set of parent nodes of nodes in $\mathcal{S}$. For a set of n edges $\mathcal{P} = \{(v_i, w_i)\}_{i=1}^n$, let $\mathcal{V}(\mathcal{P}) := \bigcup_{i=1}^n \{v_i, w_i\}$ 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 $|\mathcal{V}(\mathcal{P})| = |\bigcup_{i=1}^n \{v_i, w_i\}| = 2n$. We denote by $E(G)$ (resp., $V(G)$) the edges $\mathcal{E}$ (resp., vertices $\mathcal{V}$) of G. 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 $\{0,1\}^{\le n}$ 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., $i \| 1$).

Definition 1 (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 2 (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 $[N]$ 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

$$\mathsf{VC}(\mathcal{P}) := \min\{|\mathcal{S}| : \mathcal{S} \subseteq [N] \text{ covers } \mathcal{P}\}.$$

3.2 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 involved in the pebbling. We define the *complexity* of a pebbling configuration $\mathcal{P}$ as the size of a minimal vertex cover of $\mathcal{P}$. For a pebbling sequence $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\ell)$, we define $\mathsf{VC}(\boldsymbol{\mathcal{P}}) := \max_{i \in [L]_0} \mathsf{VC}(\mathcal{P}_i)$.

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 $\mathcal{P}_{\text{target}} = \mathrm{in}(T) = \{(i,T) \in \mathcal{E}\}$ 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 certain pebbling rules.

Reversible Pebbling. We consider the reversible edge-pebbling game from [JKK+17].

Definition 3 (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 $\mathcal{P}_\ell = \mathcal{P}_{\text{target}}$, such that for all $i \in [\ell]$ 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 4 (Node-Pebbling)**.** A node pebbling of a DAG $G = (\mathcal{V}, \mathcal{E})$ with *unique sink* T is a pebbling sequence $\boldsymbol{\mathcal{P}} = (\mathcal{P}_0, \ldots, \mathcal{P}_\ell)$ with $\mathcal{P}_0 = \mathcal{P}_{\text{start}}$ and $\mathcal{P}_\ell = \mathcal{P}_{\text{target}}$, such that for all $i \in [\ell]$ there is a unique $v \in [N]$ 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 5 (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 3 and 4, the configuration graph is undirected.

4 The Builder-Pebbler Game

In this work, we consider security games for multi-user schemes where an adversary can adaptively do the following actions:

- 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 6 (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 7 (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 6. 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 2. 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 8 (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 $X(G) = \emptyset$.

4.1 Player Strategies

Builder strategies. As motivated in Remark 1, we will be dealing in this paper mostly with a class of Builders who play *independently* of Pebbler's strategy.[13]

Definition 9 (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.

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 10 (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 $\mathcal{P}$.

[13]The exact definition of the strategy will depend on the graph and the application: e.g., see Theorem 6 for an oblivious Builder strategy for paths that is later used in Corollary 2.

Note that the notion of obliviousness differs from that in Definition 9.[14] Definition 10 is motivated by oblivious reductions used in [JKK+17] (see Section 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 10 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 reductions [JKK+17, FKKP19] that rely on the node-pebbling game from Definition 4. 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 11 (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 3. 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.

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 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 5.2). Finally, we show a bound that holds for arbitrary Pebblers (Section 5.3).

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

[14] 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.

unrestricted Builder-Pebbler Game, which allows to deduce a much stronger bound, but which may not be as widely applicable.

5.1.1 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 5.1.1) has advantage at most quasi-polynomial in N (Section 5.1.1). 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 12 (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* $\mathcal{X}$ 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 8 as the frontier of this cut.

Remark 4. A complete characterisation of such reachable configurations is given in [LTV98]. Let the pebbles in a configuration $\mathcal{P}$ be $\{(v_i, v_i + 1)\}_{i \in [\log(N)]}$ for $v_i \in$

$[0, N-1]$. Then $\mathcal{P}$ is reachable if and only if for every $i \in [\log(N)]$, $\mathcal{P}$ 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 $\mathcal{S} \subseteq [N]$ 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 $[N]$, 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 1. *Let $\mathcal{S} \subseteq [N]$ 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*

$$\mathsf{Pr}[\exists \mathcal{P} \in P : \mathcal{S} \textit{ covers } \mathcal{P} \wedge \mathcal{S} \subseteq V(\mathcal{P})] \leq \frac{s^{2s}}{N^{s-\sigma} 2^{\sigma(\sigma+1)/2}} \leq \frac{N^{\log(\log(N))}}{N^{\log(N)/8}}.$$

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 $\in [2^{\log(N)-s'}]$, another one is of size $\in [2^{\log(N)-s'+1}]$, another one of size $\in [2^{\log(N)-s'+2}]$, and so on, up to size $\in [N/2]$.[15]
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 $\in [2^{\log(N)-\sigma}]$, one of size $\in [2^{\log(N)-\sigma+1}]$, and so on, up to size $\in [2^{\log(N)-s'-1}]$. For

[15]Note, we also consider the distance between the source node and the first node in $\mathcal{S}$ on the path as a gap.

the remaining s' gaps, as stated above, there must be one gap of size $\in [2^{\log(N)-s'}]$, one of size $\in [2^{\log(N)-s'+1}]$, up to size $\in [N/2]$. Thus, independent of $\bar{s}$, there must be $s-\sigma$ gaps of size 1 and σ gaps of sizes $\in [2^{\log(N)-\sigma+k-1}]$ for $k \in [\sigma]$, 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 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 1 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 6 (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 12 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 $\mathcal{S} \subseteq [N]$ in the beginning of the game. Since B queries edges uniformly at random, $\mathcal{S}$ is a uniformly random subset of $[N]$. Thus, by Lemma 1, 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. ☐

5.1.2 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 5.1.2) has advantage at most quasi-polynomial in N (Section 5.1.1). As a warm up, we analyse in Section 5.1.2 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 5.1.2). 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 13 (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 $\mathcal{X} = \{(\mathcal{P}_i, \mathcal{P}_j) \mid \mathcal{P}_i \in \boldsymbol{\mathcal{P}} \wedge \mathcal{P}_j \notin \boldsymbol{\mathcal{P}}\}$. Note that any $\mathcal{P}_i$ with $(\mathcal{P}_i, \mathcal{P}_j) \in \mathcal{X}$ 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 8 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 2 that this cannot decrease the success probability too much, so any super-polynomial lower bound we obtain in this way holds in general.

Lemma 2. *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 7 (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 13 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 2 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 8 (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 13 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 7, by Lemma 2 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 7, the probability of $\mathsf{P}_{\alpha(s)}$ selecting a vertex cover that is minimal for a configuration that is in $\mathcal{X}$ 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

$$N^{-\log(N)+\log(\log(N/2))+o(1)}.$$

$\square$

5.1.3 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 $[N/2]$ 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 $[N/2]$ 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 $[N/2]$, 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 3 (Lower Bound on VC-complexity of Complete Graphs)**.** *Let $\boldsymbol{\mathcal{P}}_N = (\mathcal{P}_{\text{start}}, \dots, \mathcal{P}_{\text{target}})$ 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 $\mathcal{P}^*, \dots, \mathcal{P}_{\text{target}}$. 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 $\mathcal{P}^*, \dots, \mathcal{P}_{\text{target}}$ 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 $N \geq 1$. $\square$

This also yields the following definition of good configuration for the complete graph.

Definition 14 (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 $\boldsymbol{\mathcal{P}} = (\mathcal{P}_{\text{start}}, \dots, \mathcal{P})$ 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 $\mathcal{X}$ 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 8 as the frontier of this cut.

The Upper Bound. Lemma 3 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 9 (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 14 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 $S \subseteq [N]$ in the beginning of the game and B chose a permutation of $[N]$ 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. □

5.2 Node Pebbler

Here, we consider *node Pebblers* as defined in Definition 11, 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 4). As we will see in Section 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.

5.2.1 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 15. 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 define $D(G) = \max\{|E_d| \mid E_d \subset E(G) \wedge |\{u \mid (u,v) \vee (v,u) \in E_d\}| = 2|E_d|\}$ to be the maximum number of pairwise disjoint edges in G.

Theorem 10. *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 10 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

$$\binom{D(\mathcal{S}_v) + D(\mathcal{P}_v^2)}{D(\mathcal{P}_v^2)}^{-1}. \tag{1}$$

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 1. *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 10 with $v = N/2$. Note that $\mathcal{P}_v^2$ is the entire subgraph induced by $[N/2 - 1]$, and similarly $\mathcal{S}_v$ 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 10 any node Pebbler P has advantage at most

$$\pi \leq \binom{N/2}{N/4}^{-1} \leq 2^{-\Omega(N)}.$$

□

5.3 Unrestricted Pebbler

5.3.1 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 5.1.1.

Generalized Pebbling Characteristics of Paths. Let $k \in [N]$ 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 5.1.1. 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 16 (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[16], 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* $\mathcal{X}$ 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 8 as the frontier of this cut.

[16]For technical reasons, we exclude the first subpath of length k in C.

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 3).

Theorem 11 (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 12 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 16 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 $[N]$ 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 δ_{out}^{2k} sinks at depth $2k$ (see Figure 3). For all rounds such that $R > 2k$ and $R \equiv 1 \mod k$, the Builder B first chooses an integer $i \in [\delta_{out}^{k}]$ and then only queries edges outgoing from the ith batch of δ_{out}^{k} sinks at depth $R - 1$. Finally, B chooses the target node uniformly at random from the δ_{out}^{2k} sinks at depth $D = N/\delta_{out}^{2k}$ (see Figure 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 δ_{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

$$\pi \leq 1/\delta_{out}^{k \cdot \lfloor \log(\lceil D/k \rceil - 1) \rfloor}. \tag{2}$$

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 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 3.(b)). $\square$

6 Cryptographic Lower Bound I: Generalised Selective Decryption

The generalized selective decryption game (GSD) was informally introduced in Section 1. In this section, we formally define GSD and interpret the lower bounds from Sections 5.1.1 to 5.1.3 and 5.3.1 for GSD. Then, in Section 6.3, we define the public-key analogue of GSD [ACC+19], where PKE is used instead of SKE as the underlying primitive. We will establish analogous lower bounds for public-key GSD, which will serve as a basis for the lower bound on TreeKEM in Section 7.

6.1 Definition and Security Assumption

We use the definitions from [JKK+17].

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 $\mathcal{K} \subseteq \mathcal{M}$ (i.e., we can encrypt keys). We assume that $(\mathsf{Enc}, \mathsf{Dec})$ is correct, i.e.,

$$\forall \mathtt{k} \in \mathcal{K}, m \in \mathcal{M} \ : \ \mathsf{Pr}[\mathsf{Dec}(\mathtt{k}, \mathsf{Enc}(\mathtt{k}, m)) = m] = 1$$

and that it is ε-indistinguishable under chosen-plaintext attack (IND-CPA) – see Definition 17.

Definition 17 (IND-CPA)**.** The game is played between a challenger (either G_0 or G_1) and an adversary on the symmetric encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$. The challenger chooses the challenge key $\mathtt{k} \leftarrow \mathcal{K}$. The adversary can make two types of queries:

- Encryption queries $(\mathtt{encrypt}, m)$, $m \in \mathcal{M}$: the challenger returns $\mathsf{Enc}(\mathtt{k}, m)$.
- One challenge query $(\mathtt{challenge}, m_0, m_1)$, $m_0, m_1 \in \mathcal{M}$: the challenger when simulating G_b returns the challenge ciphertext $\mathsf{Enc}(\mathtt{k}, m_b)$.

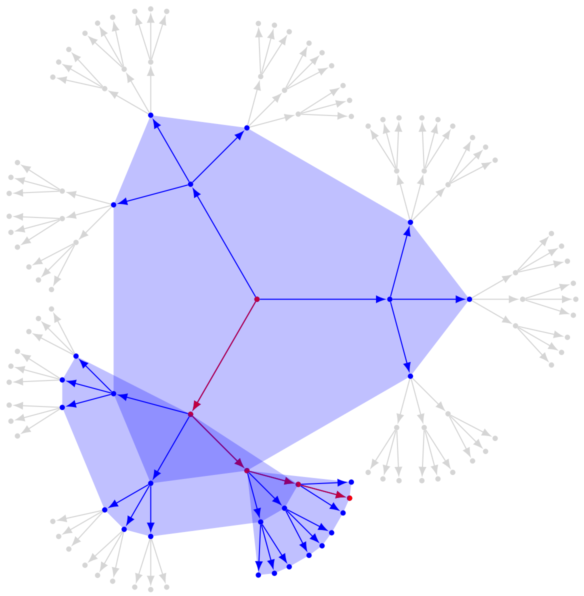
An encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ is said to be ε-indistinguishable under chosen-plaintext attack, if G_0 and G_1 are ε-indistinguishable.

The GSD game is defined as follows:

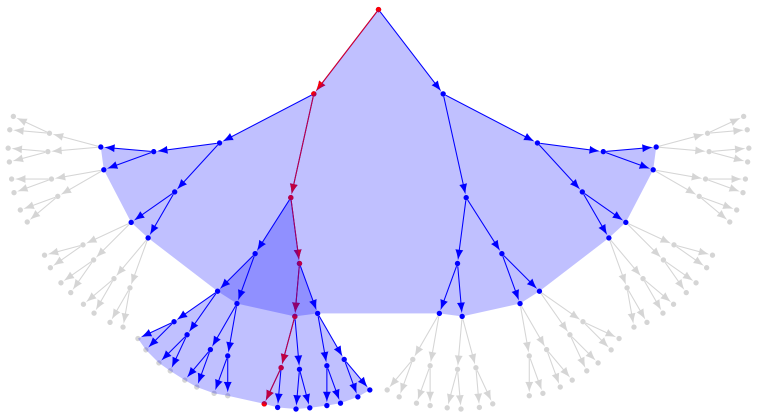
Definition 18 (Adaptive GSD [Pan07, JKK+17])**.** The game is played between a challenger G (which is either G_L or G_R) and an adversary A using an SKE scheme $(\mathsf{Enc}, \mathsf{Dec})$. G picks N keys $\{\mathtt{k}_1, \ldots, \mathtt{k}_N\} \leftarrow \mathcal{K}$ uniformly at random, and initialises a key-graph $G := (\{v_1, \ldots, v_N\}, \emptyset)$; it also initialises a set $\mathcal{C} = \emptyset$. A can make three types of queries:

- Encryption queries, $(\mathtt{encrypt}, v_i, v_j)$: G returns $c_{i,j} \leftarrow \mathsf{Enc}(\mathtt{k}_i, \mathtt{k}_j)$, and adds (v_i, v_j) to $\mathcal{E}$.[17]

[17] If A repeatedly queries encryptions of k_j under k_i, it always obtains the same encryption $c_{i,j}$. Hence, we assume A never queries an edge twice.



(a)



(b)

Figure 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 11 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 gray. 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.

- Corruption queries, $(\mathtt{corrupt}, v_i)$: G returns $\mathtt{k}_i$, and adds v_i to $\mathcal{C}$.
- One challenge query $(\mathtt{challenge}, v_i)$: Here the answer differs between G_L and G_R: G_L answers with $\mathtt{k}_i$ (real key), whereas G_R answers with $r \leftarrow \mathcal{K}$ (fake key) sampled uniformly at random — for the task to be non-trivial, v_i must be a sink and must *not* be reachable from any vertex in $\mathcal{C}$.

In the GSD game restricted to $\mathcal{G}$ (for some family of graphs on N vertices $\mathcal{G}$), the adversary is restricted to query key-graphs $G \subseteq G' \in \mathcal{G}$.

Definition 19. An SKE scheme $\mathsf{SKE} = (\mathsf{Enc}, \mathsf{Dec})$ is called ε-adaptive GSD-secure (restricted to $\mathcal{G}$) if G_L and G_R are ε-indistinguishable.

Definition 20 (Black-Box and Straight-Line GSD Reduction)**.** R is a *black-box* GSD reduction if for every SKE $\mathsf{SKE} = (\mathsf{Enc}, \mathsf{Dec})$ and every adversary A that wins the GSD game played on SKE, R breaks SKE. Moreover, if A is an (ϵ, t) GSD adversary and R (ϵ', t')-breaks SKE (where ϵ' and t' are functions of ϵ and t) 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.

6.2 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.

The following definition mirrors the obliviousness of Pebblers in the context of the Builder-Pebbler Game (cf. Definition 10).

Definition 21 (Oblivious GSD Reduction)**.** A straight-line GSD reduction R (Definition 20) 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 22 (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 `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. 6.3) 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 4 (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 *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* $\mathbf{poly}(N)/2^{\Omega(N)}$*). 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 k, the message m and the randomness r are all λ-bit long, $\mathsf{Enc}(\mathtt{k}, m; r)$ 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 GameBuilders 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 $\texttt{k}_j$ under the key $\texttt{k}_i$, 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 $\mathsf{A} \in \mathbf{PSPACE}$ 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 $\mathsf{P} = \phi(\mathsf{R})$ 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 $(\texttt{k}, m_b, r)$ such that $c^* = \mathsf{Enc}(\texttt{k}, m_b; r)$ (where c^* is the challenge ciphertext) is queried to Enc. Since R is PPT, the probability of R doing this is at most $\mathbf{poly}(N)/2^{\Omega(N)}$. Accordingly, to gain a larger advantage, R must send c^* to A as response to some edge query. Since $\mathsf{B} = \phi(\mathsf{A})$ is oblivious, the behaviour of A does not depend on 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

$$\sum_{(\mathcal{P}_i, \mathcal{P}_j) \in \mathcal{P}^{G^t}} p_{i,j} \left| \mathsf{Pr}\left[\mathsf{A}(\mathcal{P}_i) \to 1\right] - \mathsf{Pr}\left[\mathsf{A}(\mathcal{P}_j) \to 1\right] \right|$$

where $p_{i,j}$ is the probability that the query phase results in the configuration $\mathcal{P}_i$ or $\mathcal{P}_j$ depending on 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 c^* represents a consistent encryption edge (i.e. is not a pebble) and equal to $\mathcal{P}_j$ if 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 $|\mathsf{Pr}[\mathsf{A}(\mathcal{P}_1) \to 1] - \mathsf{Pr}[\mathsf{A}(\mathcal{P}_2) \to 1]| = 0$ 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 c^* is considered not a pebble for this argument). This is exactly the advantage of Pebbler $\mathsf{P} = \phi(\mathsf{R})$ 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. $\square$

The following lower bounds on GSD now easily follow from Lemma 4 and the theorems in the previous section (Theorems 6, 8, 9 and 11, resp.).

Corollary 2 (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 3 (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 9 gives the following result.

Corollary 4 (Lower Bound for GSD on Arbitrary DAGs, Oblivious Reductions)**.** *Any* key-committing oblivious *reduction proving adaptive (unrestricted) GSD-security 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 2.

Corollary 5 (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-security 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

$$\Lambda \geq N^{\log(N)/8-\log(\log(N))/4}.$$

6.3 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. It was recently introduced in [ACC+19] to analyse the security of continuous group key agreement protocols like TreeKEM [BBR18, BBM+20]. We recall the formal definition in Definition 23 and then extend the lower bounds that we established for GSD in Corollaries 2 to 5 to public-key GSD (Corollaries 6 to 9). Corollary 9 will be used later in Section 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. [ACC+19] simply sent 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 23 (Public-Key GSD [ACC+19])**.** 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 $i \in [N]$ 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 $c = \mathsf{Enc}_{\mathtt{pk}_i}(\mathtt{sk}_j)$ 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 $\mathcal{R}$.

- $(\texttt{challenge}, v_i)$, single access: On input a challenge node v_i, C samples $b \leftarrow \{0,1\}$ uniformly at random and returns $\texttt{sk}_i$ 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 (ϵ, t)-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 4, 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 20 and 21).

Lemma 5 (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* $\mathbf{poly}(N)/2^{\Omega(N)}$*). Moreover, if* R *is oblivious then so is* P*.*

Proof (Sketch). The proof of this lemma is similar to that of Lemma 4 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 4 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 5 and the Theorems 6, 8, 9 and 11, resp., in Section 5. These are analogous to Corollaries 2 to 5 for GSD. Note that oblivious reductions for public-key GSD can be defined similar to Definition 10. 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 6 (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 7 (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 8 (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 9 (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 Cryptographic Lower Bound II: (Asynchronous) Continuous Group Key Agreement

The main motivation behind the definition of GSD in [Pan07] was to analyse 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$^+$20] (and its variant Tainted TreeKEM [ACC$^+$19]) can roughly be regarded to be the public-key analogue of LKH – the security of TreeKEM (and its variants) is, in fact, analysed using the public-key GSD game (Definition 23).

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 6.3 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 2 to 5 cannot be used in this setting per se.

Nevertheless, we show in Section 7.2 that it is possible to 'emulate' some of these strategies: in particular, the Builder strategy from Theorem 11 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 10). 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 10.4).

Remark 5. In an orthogonal line of work, [MP04] showed lower bounds on the *communication complexity* of generic multicast encryption protocols. Dodis et al. [BDR20] recently build on that work to show lower bounds on the communication complexity of generic CGKA protocols.

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, ACC$^+$19] for a more formal treatment.

7.1.1 CGKA: Syntax and Security Model

CGKA is a public-key, multi-user primitive which involves M users denoted $U_0, \ldots, U_{M-1}$. Any user U_i can initialise a group $\mathcal{U}_0 \subseteq \{U_0, \ldots, U_{M-1}\}$ 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, U_i must know the (current) public key $\mathtt{pk}_j$ of each invitee $U_j \in \mathcal{U}_0$. Once a group has been created, CGKA allows any group member U_i to *update* their key using $(\mathtt{update}, U_i)$. 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 U_i can use $(\mathtt{add}, U_i, U_j)$ to *add* a new user U_j, and use $(\mathtt{remove}, U_i, U_j)$ to *remove* an already-existing member U_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 $\mathtt{K}_0, \ldots, \mathtt{K}_Q$.[18]

Handling asynchronicity. The *group operations* – $\mathtt{update}$, $\mathtt{add}$ and $\mathtt{remove}$ – 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., $(\mathtt{remove}, U_i, U_j)$ and $(\mathtt{remove}, U_j, U_i)$). This is handled in CGKA via the $\mathtt{confirm}$ operation: $(\mathtt{confirm}, U_i, g_i)$ is used to confirm or reject U_i's request for a group operation g_i (U_i then proceeds to update their own local state

[18]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.

accordingly). In case the group operation g_i is confirmed, the server delivers the message to the members and a member $U_j \neq U_i$ uses $(\texttt{process}, U_j, g_i)$ 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. This is captured formally by the following adversarial model.

Definition 24 (Adaptive Security of CGKA [ACC+19])**.** The security for CGKA is modelled using a game between a challenger C and an adversary A. At the beginning of the game, A creates a group $\mathcal{U}_0 \subseteq \{U_0, \ldots, U_{M-1}\}$ by making a query $(\texttt{initialise}, U_i, \mathcal{U}_0)$. A can then make a sequence of Q queries, enumerated below, in any arbitrary order. On a high level, the first four items allow the adversary to carry out any group and delivery operations of its choice, whereas `start-corrupt` and `end-corrupt` enable it to corrupt any user 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. $(\texttt{add/remove}, U_i, U_j)$: U_i requests to add/remove U_j to/from the group.
2. $(\texttt{update}, U_i)$: U_i requests to refresh their current local state.
3. $(\texttt{confirm}, q, \beta)$: the q-th query in the game, which must be a group operation $g \in \{(\texttt{add}, U_i, U_j), (\texttt{remove}, U_i, U_j), (\texttt{update}, U_i)\}$ initiated by some user U_i, is either confirmed (if $\beta = 1$) or rejected (if $\beta = 0$) to U_i.
4. $(\texttt{process}, q, U_j)$: if the q-th query is as above – i.e., a group operation g initiated by U_i – then C delivers the message to U_j, who immediately processes it.
5. $(\texttt{start-corrupt/end-corrupt}, U_i)$: starts/ends leakage of the entire internal state and randomness of U_i to the adversary from this time point.
6. $(\texttt{challenge}, q^*)$: A picks a query $q^* \in [0, Q]$ corresponding to a group operation g or the initialization (if $q^* = 0$). Let $\mathtt{K}^0 = \mathtt{K}_{q^*}$ denote the group key that is sampled during this operation and $\mathtt{K}^1$ be a random, independent key. The challenger tosses a coin b and – if the challenge is *non-trivial* – the key $\mathtt{K}^b$ is given to the adversary (otherwise the game is aborted).

Roughly speaking, the challenge is non-trivial if the group key $\mathtt{K}_{q^*}$ is not efficiently derivable from the information learned from user corruption (a formal definition, which uses the notion of *safe predicates*, can be found in [ACC+19]). At the end of the game, the adversary outputs a bit b' and wins if $b' = b$. We call a CGKA scheme (Q, ϵ, t)-CGKA-secure if for any adversary A making at most Q queries and running in time t the *advantage* in guessing b is at most ϵ.

7.1.2 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 U_i 'proposes' a group operation g_i and another member U_j (potentially same as U_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 U_j in one go. We assume that the member proposing a group operation g_i (i.e., U_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 U_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 U_i. This will result in a simpler key-graph at the cost of a slightly more inefficient 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

   (b) that the leaf $i \in \{0,1\}^m$ in the ratchet tree is *reserved* for the user U_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 [ACC+19] 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 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 `add` operation later in the section. The vertices have the following semantics[19]:

[19]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

- Members are associated with leaves: the leaf $i \in \{0,1\}^m$ of member $U_i \in \mathcal{U}_q$ 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 $U_i \in \mathcal{U}_q$ 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 $U_i \in \mathcal{U}_q$ consists of the ciphertexts $\mathsf{Enc}(\mathtt{pk}_u, \mathtt{sk}_v)$ for every edge $(u,v) \in E(P_i)$ (see Figure 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).

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 U_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 (Algorithm 1) will employ (it also crucially uses the *delivery* operations though).

- *Update.* To update themself (see Figure 4.(b)), a user $U_i \in \mathcal{U}_q$ 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

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 $v^{(q+1)}$.

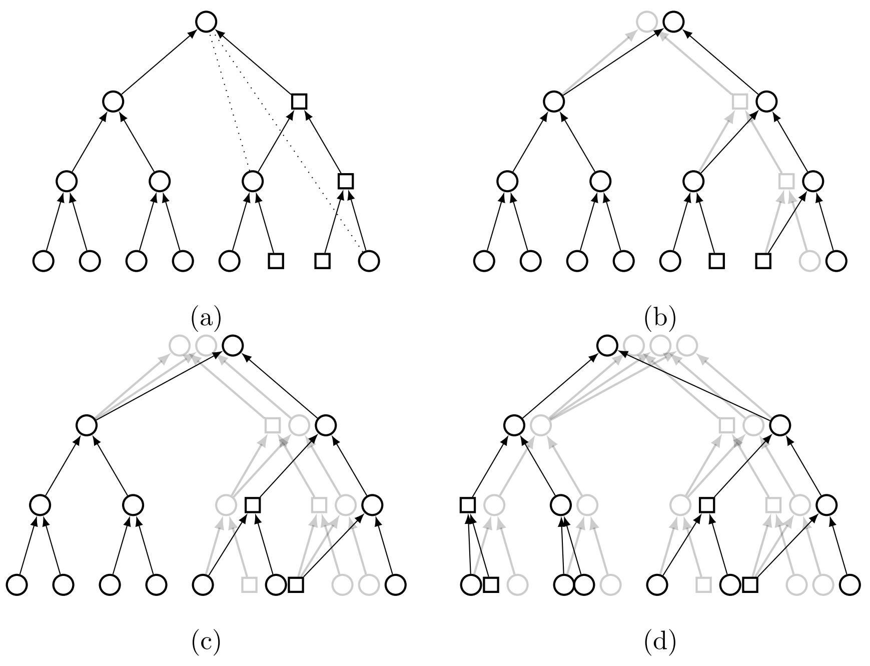


Figure 4: Ratchet tree and group operations in TreeKEM. (a) *The ratchet tree of the group in epoch q.* The blank (resp., normal) vertices are shown as squares (resp., circles). The group currently has six members $\mathcal{U}_q = \{U_0, \ldots, U_4, U_7\}$ assigned the leaves $000, \ldots, 100$ and 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 $\mathsf{Enc}(\mathtt{pk}_{10}, \mathtt{K}_q)$ (resp., $\mathsf{Enc}(\mathtt{pk}_{111}, \mathtt{K}_q)$) 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 U_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 U_5 is added to the group by U_7.* A blank path from the leaf $110'$ to the root (ε'') is added after which U_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 U_7 would know a secret key associated to a node on the path from U_4/U_5 to the root. However further nodes on the path are normal as these also lie on the path from U_7 to the root which lies outside its own path to the root. (d) *The ratchet tree after U_1 is removed from the group by U_3.* A blank path from the leaf $001'$ to the root (ε''') is added after which U_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).

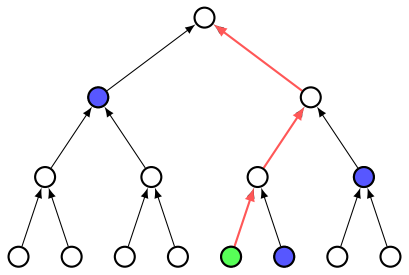


Figure 5: The path $P_{100} = 100 \to 10 \to 1 \to \varepsilon$ is highlighted in red. Its co-path vertices – 101, 11 and 0 – are also highlighted (in blue).

process this update, it adds edges from all 'co-path' vertices (see Figure 5) of P_i to P'_i, i.e.,

$$\{v_0 \dots v_{l-1}\bar{v}_l : v = v_0, \dots, v_{l-1}v_l \in V(P_i)\},$$

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), U_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 4.(b)).

- *Add and remove.* Suppose that a member $U_i \in \mathcal{U}_q$ wants to add a user U_j to the group. According to the protocol specification U_i does the following (see Figure 4):

  1. Perform an update operation 'on behalf of' U_j but using *blank* internal vertices: i.e., U_i adds a path P'_j that has a normal leaf j' (associated with the key-pair of U_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 U_i samples keys on behalf of U_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 4 they have been clubbed together into one step. The steps for a member $U_i \in \mathcal{U}_q$ to remove another member $U_j \in \mathcal{U}_q$ is mostly similar to that in `add` (see Figure 4.(d)):

1. Perform an update operation 'on behalf of' U_j but using *only blank* vertices: i.e., U_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 U_i commits a group operation g and the group enters a new epoch, they have to – in addition to the protocol message of the base protocol for g – 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 g. Moreover, another member U_j processes g 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 [ACC+19] with a quasi-polynomial (in $M = 2^m$) loss of security, which translates to the following theorem for TreeKEM.

Theorem 12 (Theorem 2 in [ACC+19] 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 $O(M^2 Q^{O(m)})$, where $M = 2^m$ is the number of users and Q is the number of queries.*

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 7.1.2. 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 11 (via Lemma 5 and Corollary 9), 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 1, and below we provide an intuitive overview.

Overview of the query strategy. Recall that the Builder strategy B in Theorem 11 (with the overlap parameter $k = 1$) is to construct, level by level, a regular tree T_Δ with in-degree 1, out-degree (denoted here by) Δ 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_Δ into the TreeKEM key-graph (see Figure 6) with the root $v_{0,1}$ of T_Δ set as the user U_0. Before describing the query strategy of our TreeKEM adversary A, we make two observations about the TreeKEM protocol (Section 7.1.2) and the attack model (Definition 24) 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 U_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 4). We say that U_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 U_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'.[20]

With the observations in mind, our approach to simulate B is as follows (the value of Δ and D in terms of M will be specified later in Corollary 10):

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 6.(a)). To embed the level 1, A updates a set $\mathcal{U}_{0,1}$ of Δ users such that each user $U_i \in \mathcal{U}_{0,1}$ 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

[20] 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.

level-0 updates result in a set of level-1 vertices $\mathcal{V}_1 := \{v_{1,0}, \ldots, v_{1,\Delta-1}\}$, which A extends in Item 2 (see Figure 6.(b) and Figure 6.(c)).

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 Δ members $\mathcal{U}_{1,i}$ such that each member $U_j \in \mathcal{U}_{1,i}$ extends $v_{1,i}$. Then it *forks* the state of the group as follows:

   (a) U_j *processes* all messages U_i processed so that they share the same group state,

   (b) both U_j and U_i process the update of U_i (which created node $v_{1,i}$), and

   (c) U_j extends $v_{1,i}$ by updating themself.

   At the end of these level-1 updates, A will have embedded a set of Δ^2 level-2 vertices $\{v_{2,0}, \ldots, v_{2,\Delta^2-1}\}$ (see Figure 6.(d)).

3. As in B, A randomly selects a Δ-sized batch of level-2 vertices

$$\mathcal{V}_2 = \{v_{2,d_1^*\Delta}, \ldots, v_{2,(d_1^*+1)\Delta-1}\} \subset \{v_{2,0}, \ldots, v_{2,\Delta^2-1}\},$$

   where $d_1^* \in [0, \Delta - 1]$, 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_Δ: 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 $\{v_{\ell,0}, \ldots, v_{\ell,\Delta^2-1}\}$; and

   (b) a random batch $\mathcal{V}_\ell = \{v_{\ell,d_{\ell-1}^*\Delta}, \ldots, v_{\ell,(d_{\ell-1}^*+1)\Delta-1}\} \subset \{v_{\ell,0}, \ldots, v_{\ell,\Delta^2-1}\}$ 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 1 and an example of the resulting graph structure is shown in Figure 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 $v_{1,\Delta}$, 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 $U_j \in \mathcal{U}_{0,1}$). 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 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 not 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).

```
1:  $(\texttt{initialise}, U_{M-1}, \{U_0, \ldots, U_{M-1}\})$   $\triangleright$ $U_{M-1}$ fully populates the group, $U_i$ assigned leaf $i$
2:  for each $i \in \{0,1\}^m$ do   $\triangleright$ All users process the group initialisation
3:      $(\texttt{process}, 0, U_i)$
4:  end for
                                        $\triangleright$ $\mathcal{U}_{0,1}$ set as $\{U_{\Delta^2}, \ldots, U_{\Delta^2+\Delta-1}\}$
5:  for $d \in [0, \Delta-1]$ do   $\triangleright$ Each user $U_d \in \mathcal{U}_{0,1}$...
6:      $(\texttt{update}, U_{\Delta^2+d})$: query $\texttt{update}_{0,d}$   $\triangleright$ ...updates themself to extend root $0^m$...
7:  end for   $\triangleright$ ...and generate level-1 vertices $\mathcal{V}_1 = \{v_{1,0}, \ldots, v_{1,\Delta-1}\}$
8:  Set $d_0^* := 0$   $\triangleright$ Challenge path $v_{0,d_0^*} \to \ldots \to v_{D,d_D^*}$ initiated at the leaf $0^m$
9:  for $\ell \in [1, m-2\delta]$ do   $\triangleright$ For each higher level $\ell$ of $T_\Delta$
10:     for $k \in [0, \Delta-1]$ do   $\triangleright$ Fork the group for each extendable $v_{\ell, d_{\ell-1}^*\Delta+k} \in \mathcal{V}_\ell$
            $\triangleright$ Set $\mathcal{U}_{\ell,k} := \{U_{2^\ell\Delta^2+k\Delta}, \ldots, U_{2^\ell\Delta^2+(k+1)\Delta-1}\}$ and $U_i := U_{2^{\ell-1}\Delta^2+d_{\ell-1}^*\Delta+k}$
11:         for each $d \in [0, \Delta-1]$ do   $\triangleright$ For each user $U_j = U_{2^\ell\Delta^2+k\Delta+d} \in \mathcal{U}_{\ell,k}$
12:             for $l \in [0, \ell-2]$ do   $\triangleright$ $U_j$ processes all messages that...
13:                 $(\texttt{process}, \texttt{update}_{l, d_l^*\Delta}, U_{2^\ell\Delta^2+k\Delta+d})$   $\triangleright$ ...$U_i$ processed...
14:             end for   $\triangleright$ ...and both $U_i$ and $U_j$ end up in the same state
15:             $(\texttt{process}, \texttt{update}_{\ell-1, d_{\ell-1}^*\Delta+k}, U_{2^{\ell-1}\Delta^2+d_{\ell-1}^*\Delta+k})$   $\triangleright$ Both $U_i$ and...
16:             $(\texttt{process}, \texttt{update}_{\ell-1, d_{\ell-1}^*\Delta+k}, U_{2^\ell\Delta^2+k\Delta+d})$   $\triangleright$ ...$U_j$ process $U_i$'s update
17:             $(\texttt{update}, U_{2^\ell\Delta^2+k\Delta+d})$: query $\texttt{update}_{\ell, k\Delta+d}$   $\triangleright$ $U_j$ extends $v_{\ell, d_{\ell-1}^*\Delta+k}$
18:         end for   $\triangleright$ Added $v_{\ell+1,k\Delta}, \ldots, v_{\ell+1,(k+1)\Delta-1}$
19:         Sample $d_\ell^* \leftarrow [0, \Delta-1]$   $\triangleright$ Part of level $\ell$ to be extended in iteration $\ell+1$
20:     end for
21: end for
22: $(\texttt{challenge}, \texttt{update}_{m-2\delta, d_{m-2\delta}^*})$   $\triangleright$ Challenge epoch $q^*$
23: for $\ell \in [1, m-2\delta-1]$ do   $\triangleright$ For each (but extreme) level $\ell$ of $T_\Delta$ indirectly...
24:     for $d \in [0, \Delta-1] \setminus \{d_{\ell-1}^*\}$ do $\triangleright$ ...corrupt the unextended level-$\ell$ vertices by...
25:         $(\texttt{start-corrupt}, U_{2^\ell\Delta^2+d_l^*\Delta+d})$   $\triangleright$ ...corrupting the user that generated it
26:         $(\texttt{end-corrupt}, U_{2^\ell\Delta^2+d_l^*\Delta+d})$
27:     end for
28: end for
```

Algorithm 1: Query strategy for the TreeKEM adversary A, parametrised by the number of users $M = 2^m$ and degree of embedding $\Delta = 2^\delta$. Only details pertaining to the embedding of T_Δ has been included.

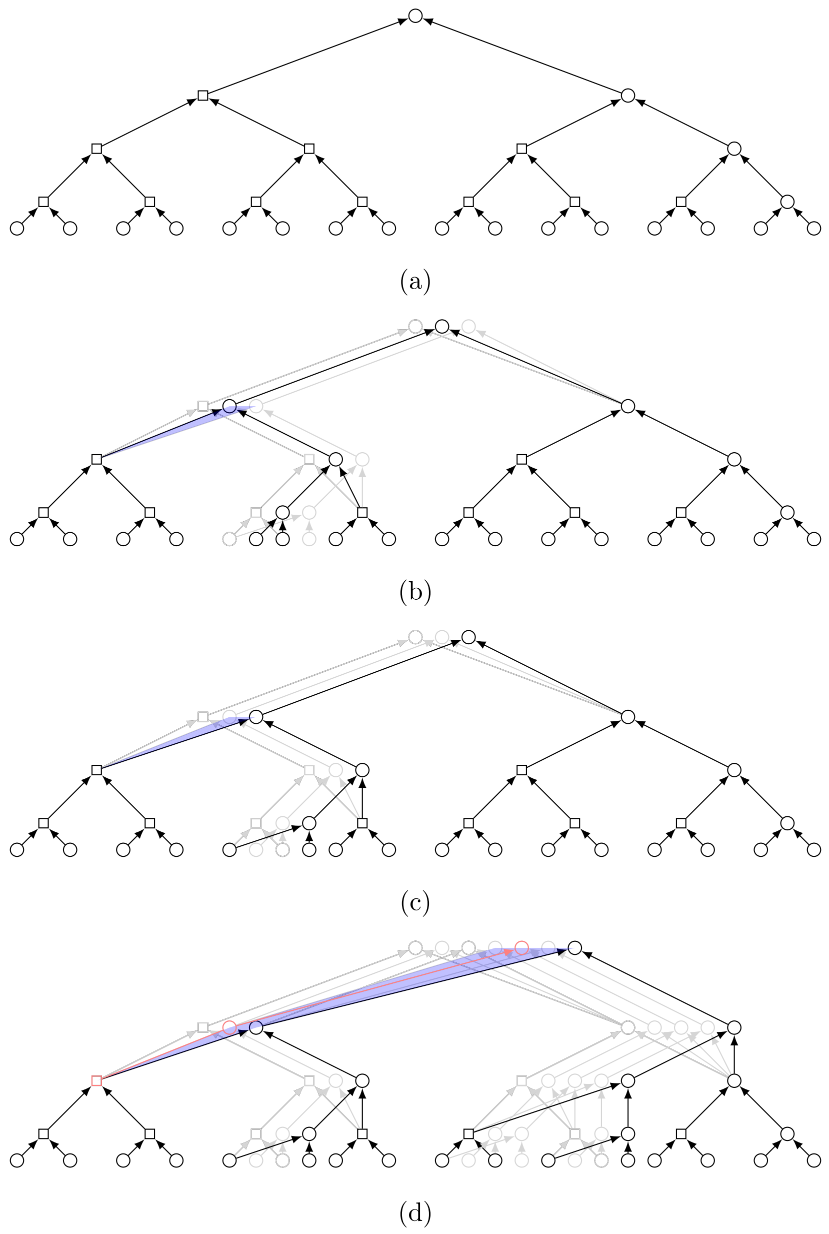


Figure 6: Embedding a regular tree of outdegree $\Delta = 2$, depth $D = 2$ and overlap $k = 1$ in the TreeKEM key-graph. (a) *Initial ratchet tree.* The ratchet tree after the group was initialised by U_{M-1} and all invitees process this initialisation. (b) and (c) *The key-graph after level-*0 *updates.* The key-graph after U_4 on leaf 0100 and U_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 U_4 and U_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 $\mathcal{U}_{1,0} = \{U_4, U_5\}$). (d) *The key-graph after level-*1 *updates.* The point of view is of U_{11} on leaf 1011. The embedding is complete (with $\mathcal{U}_{2,0} = \{U_8, U_9\}$ and $\mathcal{U}_{2,1} = \{U_{10}, U_{11}\}$): 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 7.1.2. The challenge path is highlighted in red.

The lower bound. In Lemma 6 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 10, follows once the parameters are set appropriately.

Lemma 6 (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 11. 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 the 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 $\mathbf{poly}(M)/2^{\Omega(M)}$).*

Proof (Sketch). The proof proceeds similar to Lemma 5. The ideal PKE is defined exactly as in Lemma 5 (with $\lambda = \Theta(M)$) and the query strategy of the adversary A is defined in Algorithm 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 5 to end up with a Builder-Pebbler Game. Since Algorithm 1 embeds a Δ-regular tree T_Δ of depth m in the TreeKEM key-graph G, it follows that the corresponding Builder B obtained by mapping Algorithm 1 using ϕ also ends up building a graph G such that T_Δ is embedded in it. The proof now follows by the observation that the bound we established for Lemma 5 can be extended to any graph that has T_Δ embedded in it and because the order in which the embedding is carried out in G is exactly as by the Builder in Lemma 5.

□

Corollary 10 (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 $k = 1$, $\Delta = M^{1/4}$ (denoted δ_{out} there) and $D = m/2$ in Equation (2) in the proof of Theorem 11 to arrive at an upper bound of $\pi \leq 1/M^{O(\log(m))}$ on the Pebbler's advantage. This proves the claim. □

Remark 6. It is possible to slightly improve on the constant factors in the exponent in Corollary 10 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 [ACC$^+$19] stated in Theorem 12 and our lower bound (see Table 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.

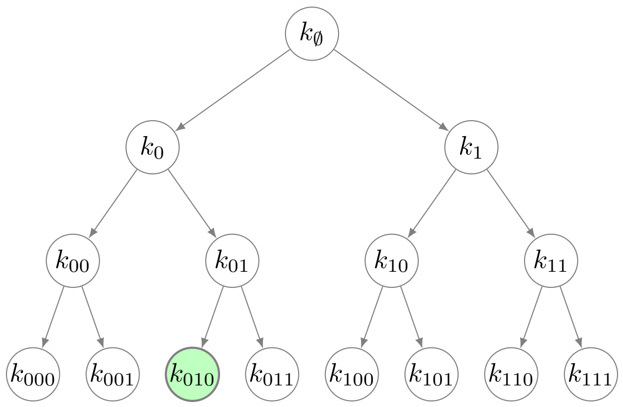


Figure 7: Illustration of the GGM PRF. Every left child $k_{x\|0}$ of a node k_x is defined as the first half of $\mathsf{PRG}(k_x)$, the right child $k_{x\|1}$ as the second half. The thick node (shaded in green) corresponds to $\mathsf{F}_{GGM}(k_\emptyset, 010)$.

8 Cryptographic Lower Bound III: Constrained PRF

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].

8.1 Definition, Construction and Security Assumption

The following definitions are essentially taken from [JKK+17].

Definition 25 (GGM PRF)**.** Given a $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$, the PRF $\mathsf{F}_{GGM} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$ is defined as

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

A graphical representation of the GGM construction is depicted in Figure 7.

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

Definition 26 (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 $k \in \mathcal{K}$, $x \in \{0,1\}^{\leq n}$ and $k_x \leftarrow \mathsf{F.Constrain}(k,x)$ satisfy

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

That is, $\mathsf{F.Constrain}(k, x)$ outputs a key k_x that allows evaluation of $\mathsf{F}(k, \cdot)$ 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 $k \leftarrow \mathcal{K}$ and $x \in \{0,1\}^l$ with $l \leq n$ defining $\mathsf{F}_{GGM}.\mathsf{Constrain}(k, x) = (k_x^1, k_x^2) := (x, \mathsf{F}_{GGM}(k, x))$ and

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

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

Definition 27. 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 $k \leftarrow \mathcal{K}$, and initialises a set $\mathcal{X} = \emptyset$. A can make at most $q = q(n)$ queries, which is either:

- Constrain queries, $(\texttt{constrain}, x)$: G returns $\mathsf{F.Constrain}(k, x)$, and adds x to $\mathcal{X}$.
- One challenge query $(\texttt{challenge}, x^*)$: Here the answer differs between G_L and G_R: G_L answers with $\mathsf{F.Eval}(k, x^*)$ (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 $\mathcal{X}$.

Definition 28. A prefix-constrained PRF F is (s, ε, q)-adaptive-secure if G_L and G_R are (s, ε)-indistinguishable.

8.2 Lower Bound for the GGM CPRF

To prove a lower bound for GGM, we use the combinatorial upper bound from Section 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 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 8). 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 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 5.3 still holds for Builders revealing such bunches of queries at once.

Lemma 7 (Coupling Lemma for GGM CPRF)**.** *Let $\mathcal{G}$ be the family of trees of depth D, size $N = \mathbf{poly}(D)$, indegree 1, outdegree 2 and a single source; i.e. $\mathcal{G}$ denotes the set of $\mathbf{poly}(D)$-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 11. 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 $\mathbf{poly}(D)/2^{\Omega(D)}$).*

To prove this lemma, we will use the following construction of an information-theoretically secure PRG scheme.

Lemma 8. *Let $\pi_0, \pi_1 : \{0,1\}^\lambda \to \{0,1\}^\lambda$ be two random permutations. Then $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ defined by $\mathsf{PRG}(x) := (\pi_0(x), \pi_1(x))$ is a $\mathbf{poly}(\lambda)/2^{\lambda/2}$-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 $\mathbf{poly}(\lambda)/2^{\lambda/2}$-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 7.

Proof of Lemma 7. We pick the PRG from Lemma 8 for $\lambda = \Theta(D)$.

Analogously to the proof of Lemma 4 we define a map ϕ between the CPRF game and the Builder-Pebbler Game:

- For a constrain query by adversary A, $(\mathtt{constrain}, x)$, 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 $x^2 = (x^2_1, x^2_2, \dots)$, it then queries the edges between the nodes associated with x^1 and $x^1||x^2_1$, between $x^1||x^2_1$ and $x^1||x^2_1||x^2_2$, 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 11 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 11 as follows: A first queries CPRF evaluations for $\{0,1\}^{2k}||0^{D-2k+1}$ in reverse order (i.e. starting from $1^{2k}||0^{D-2k+1}$)[21] – this is in analogy to the first $2k$ rounds of B (see Figure 8). Then it proceeds in $[D/k-2]$ rounds, where in round $j \in [D/k-2]$ it first samples $x_j^* \in \{0,1\}^k$ and then makes 2^{2k} queries $x_1^*||\ldots||x_j^*||\{0,1\}^{2k}||0^{D-(j+2)k+1}$ in reverse order, starting with $x_1^*||\ldots||x_j^*||1^{2k}||0^{D-(j+2)k+1}$. 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 $j \in [D]$. 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

[21]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}.

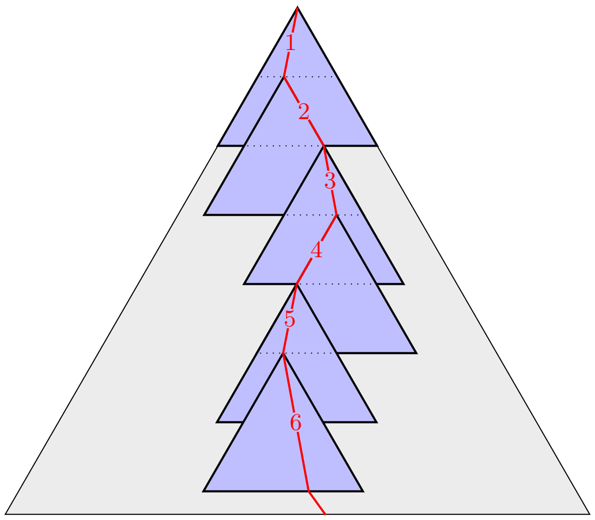


Figure 8: A schematic diagram showing the adversarial query strategy for GGM CPRF in Lemma 7. The outer (gray) triangle represents the perfect binary tree of depth $D = 7k + 1$ (also see Figure 3) representing the GGM PRF. The internal (blue) triangles represent perfect binary trees of depth $2k$ with 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^*.

challenge seed is negligibly small $(\mathbf{poly}(D)/2^{\Omega(D)})$. 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 4, 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.

$\square$

Using the above lemma, the following corollary now easily follows from Theorem 11.

Corollary 11 (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*

$$\Lambda \geq n^{\log(n)/2-\log(\log(n))/2}.$$

9 Cryptographic Lower Bound IV: Proxy Re-encryption

As an application of our results on node Pebblers, we consider the recent work by Fuchsbauer et al. [FKKP19] on adaptively-secure proxy re-encryption (PRE). In particular, they identify two natural security properties – indistinguishability of ciphertexts and δ-weak key privacy – which allow them 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 current work now allows 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 7. Other applications considered by Jafargholi et al. [JKK+17], such as Secret Sharing and Yao's Garbled Circuit, 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 from [JKK+17], 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. (As mentioned in Section 1.2.1, [KKPW] managed to show lower bounds for Yao's Garbled Circuit.)

9.1 Definitions and Security Assumptions

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 $\mathtt{pk}'$ [BBS98]. 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 below – we use the definitions and security assumptions from [FKKP19].

Definition 29 (Multi-hop, unidirectional PRE [FKKP19])**.** A multi-hop, unidirectional PRE scheme for a message space $\mathcal{M}$ consists of the six-tuple of algorithms (S, $\mathsf{K}, \mathsf{RK}, \mathsf{E}, \mathsf{D}, \mathsf{RE}$), which are explained below.

$\mathsf{S}(1^\lambda, 1^\nu) \to \mathtt{pp}$**:** On input the security parameter λ and the maximum level ν (both in unary) supported by the scheme, **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) $\mathtt{rk}_{i,j}$.

$\mathsf{E}(\mathtt{pk}, (m, \ell)) \to (c, \ell)$**: Encryption** takes as input the public key $\mathtt{pk}$, a message m and a level $\ell \in [\nu]$, and outputs a level-ℓ ciphertext (c, ℓ).

$\mathsf{D}(\mathtt{sk}, (c, \ell)) \to m$: On input a ciphertext (c, ℓ) and the secret key $\mathtt{sk}$, **decryption** outputs a message m, or the symbol $\perp$ (if the ciphertext is invalid).

$\mathsf{RE}(\mathtt{rk}_{i,j}, \mathtt{pk}_i, \mathtt{pk}_j, (c_i, \ell)) \to (c_j, \ell + 1)$**: 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 c_i under $\mathtt{pk}_i$ and transforms it to a level-$(\ell + 1)$ ciphertext c_j under $\mathtt{pk}_j$. Only ciphertexts belonging to levels $\ell \in [\nu - 1]$ can be re-encrypted.

Definition 30 (Correctness [ABH09, FKKP19]). A proxy re-encryption scheme (as in Definition 29) is correct w.r.t. the message space $\mathcal{M}$ if the following two properties hold:

1. *Correctness of encryption:* $\forall \lambda, \nu \in \mathbb{N}\ \forall\, \mathtt{pp} \in [\mathsf{S}(1^\lambda, 1^\nu)]\ \forall\, (\mathtt{pk}, \mathtt{sk}) \in [\mathsf{K}(\mathtt{pp})]\ \forall\, (m, \ell) \in \mathcal{M} \times [\nu]$:
$$\mathsf{Pr}\left[\mathsf{D}\big(\mathtt{sk}, \mathsf{E}(\mathtt{pk}, (m, \ell))\big) \neq m\right] = \mathbf{negl}(\lambda, \nu),$$
where the probability is over the random coins of E.

2. *Correctness of re-encryption:* $\forall\, \lambda, \nu \in \mathbb{N}\ \forall\, \mathtt{pp} \in [\mathsf{S}(1^\lambda, 1^\nu)]\ \forall\, (\mathtt{pk}_i, \mathtt{sk}_i)$, $(\mathtt{pk}_j, \mathtt{sk}_j) \in [\mathsf{K}(\mathtt{pp})]\ \forall\, \mathtt{rk}_{i,j} \in [\mathsf{RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)]\ \forall\, (m, \ell) \in \mathcal{M} \times [\nu - 1]$:
$$\mathsf{Pr}\left[\mathsf{D}\big(\mathtt{sk}_j, \mathsf{RE}(\mathtt{rk}_{i,j}, \mathtt{pk}_i, \mathtt{pk}_j, (c_i, \ell))\big) \neq m\right] = \mathbf{negl}(\lambda, \nu),$$
where (c_i, ℓ) is a level-ℓ ciphertext of m under $\mathtt{pk}_i$ resulting either from direct encryption or a reencryption of level-$\ell - 1$ ciphertext, and the probability is over the random coins of E and RE.

We consider the CPA-security from [FKKP19] as defined in Game 1. 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 (see Figure 9): 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).[22]

[22]In [FKKP19], 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.

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[23], 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 in Figure 9, this corresponds to the requirement that the challenge node is not reachable from any corrupt node.

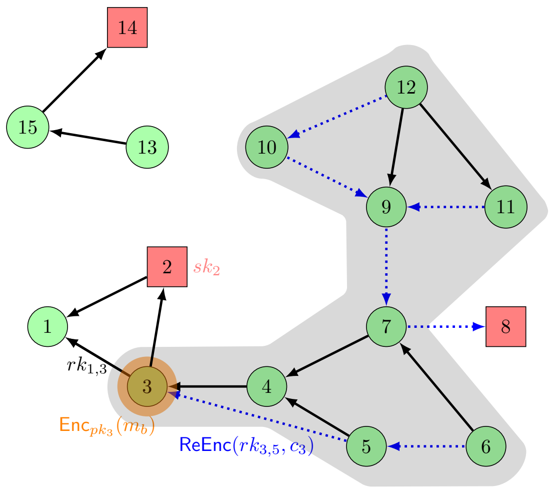


Figure 9: Recoding and challenge graph as generated in game CPA^b. The round green nodes represent the honest users, the square red ones the corrupted users. Solid black arrows from node i to node j represent that a rekey $rk_{j,i}$ from j to i was issued, and, similarly, dotted blue arrows represent re-encryption queries. For the definition of the recoding graph we do not distinguish between these two types of edges. The challenge node is marked in orange, here node 3, and the challenge graph (shaded grey) is the subgraph induced on all the ancestors of the challenge node.

Definition 31 (PRE-CPA-security [FKKP19])**.** A PRE scheme is ϵ-adaptively secure against chosen-plaintext attack if there is no adversary which can distinguish CPA^0 from CPA^1 with advantage larger than ϵ, where CPA^b is defined in Game 1.

Fuchsbauer et al. [FKKP19] reduce the CPA security of a PRE scheme to the following two basic security properties which are naturally satisfied by the popular constructions they analysed. The first basic security property is indistinguishability of ciphertexts, as defined for public-key encryption in [GM82], but on all levels:

Definition 32 (Indistinguishability)**.** A proxy re-encryption scheme PRE has ϵ-indistinguishable ciphertexts if no adversary can distinguish IND^0 from IND^1 with advantage larger than ϵ, with IND as in Game 2.

[23]In [FKKP19], the authors also consider 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. They prove 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.

```
Challenger $\mathsf{CPA}^b(1^\lambda, 1^\nu, n)$
1: Set $\mathcal{C}, \mathcal{E} = \emptyset$    ▷ Stores corrupt keys and issued re-keys and re-encryptions
2: $\mathsf{pp} \leftarrow \mathsf{PRE.S}(1^\lambda, 1^\nu)$, $(\mathtt{pk}_1, \mathtt{sk}_1), \dots, (\mathtt{pk}_n, \mathtt{sk}_n) \leftarrow \mathsf{PRE.K}(\mathsf{pp})$    ▷ Generate keys
3: $\forall i, j \in [n], i \neq j : \mathtt{rk}_{i,j} \leftarrow \mathsf{PRE.RK}((\mathtt{pk}_i, \mathtt{sk}_i), \mathtt{pk}_j)$    ▷ Generate re-keys
4: $b' \leftarrow \mathsf{A}^{(\mathtt{corrupt},\cdot),(\mathtt{rekey},\cdot,\cdot),(\mathtt{reencrypt},\cdot,\cdot,\cdot),(\mathtt{challenge},\cdot,\cdot,\cdot)}(\mathsf{pp}, \mathtt{pk}_1, \dots, \mathtt{pk}_n)$
5: if $\mathsf{A}$ made call $(\mathtt{challenge}, i^*, \cdot, \cdot)$ for some $i^*$ then    ▷ Check for abort
6:     if $\exists\, i \in \mathcal{C} : i$ is connected to $i^*$ in $([n], \mathcal{E})$ then return $0$ end if
7: end if
8: return $b'$

Oracle $(\mathtt{corrupt}, i)$
1: Add $i$ to $\mathcal{C}$
2: return $\mathtt{sk}_i$

Oracle $(\mathtt{rekey}, i, j)$
1: Add $(j, i)$ to $\mathcal{E}$    ▷ Add to recoding graph
2: return $\mathtt{rk}_{i,j}$

Oracle $(\mathtt{reencrypt}, i, j, (c_i, \ell))$
1: Add $(j, i)$ to $\mathcal{E}$    ▷ Add to recoding graph
2: return $(c_j, \ell + 1) \leftarrow \mathsf{PRE.RE}(\mathtt{rk}_{i,j}, \mathtt{pk}_i, \mathtt{pk}_j, (c_i, \ell))$

Oracle $(\mathtt{challenge}, i^*, (m_0^*, m_1^*), \ell^*)$    ▷ Single access
1: return $(c_{i^*}, \ell^*) \leftarrow \mathsf{PRE.E}(\mathtt{pk}_{i^*}, (m_b^*, \ell^*))$
```

Game 1: PRE-CPA[FKKP19]

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

Oracle $(\mathtt{challenge}, (m_0^*, m_1^*), \ell^*)$
1: return $\mathsf{PRE.E}(\mathtt{pk}, (m_b^*, \ell^*))$
```

Game 2: Security game IND for ciphertext indistinguishability

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

Game 3: Security game KP for weak key-privacy [FKKP19]

The second security property is δ-weak key privacy, which says that a set of δ re-encryption keys $\mathtt{rk}_{0,i}$ from a given source key $(\mathtt{pk}_0, \mathtt{sk}_0)$ to δ given target public keys $\mathtt{pk}_i$, where $i \in [\delta]$, is indistinguishable from a set of δ rekeys which were generated from a freshly sampled source key pair $(\mathtt{pk}_0', \mathtt{sk}_0')$. The security game for weak key-privacy as considered in [FKKP19] is given in Game 3 where the simulator RK^* is defined as

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

Definition 33 (Weak key-privacy [FKKP19]). Let $\delta \in \mathbb{N}$. A proxy re-encryption scheme PRE is (ϵ, δ)-weakly key-private if no adversary can distinguish KP^0 from KP^1 with advantage larger than ϵ, where KP is defined in Game 3.

9.2 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 [FKKP19], 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 34 (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 35 (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 9 (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*

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 $\mathbf{poly}(N)/2^{\Omega(N)}$). Moreover, if R is oblivious, then so is P.*

Proof. The proof is analogous to the one of Lemma 4, 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 6.3, 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 4, 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 $m_0 \neq m_1$. 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 m_0 or m_1 under the correct key. If the encrypted message is m_0 (resp. 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 4, with the δ-key privacy challenge taking the role of the IND-CPA challenge. □

Corollary 12 (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 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 13 (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 1 implies an exponential lower bound on the security loss even for non-oblivious black-box reductions:

Corollary 14 (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 10 does not hold (and thus neither Corollary 1) 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 14 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 (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.

10 Open Questions

We conclude this work by explaining some of the open questions and avenues for further improving our results.

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 14), 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 6. 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 $e = (u, v) \leftarrow \mathcal{E} := [N]_0 \times [N]_0 \setminus \{(x, x) \mid x \in [N]_0\}$, 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).

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 10.1 it is easy to see that our lower bound established in Corollary 11 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.

10.3 Better Reductions for Other Graph Families

We showed in Section 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.

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. 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.

References

[ABH09] Giuseppe Ateniese, Karyn Benson, and Susan Hohenberger. Key-private proxy re-encryption. In Marc Fischlin, editor, *CT-RSA 2009*, volume 5473 of *LNCS*, pages 279–294. Springer, Heidelberg, April 2009. (Cited on page 60.)

[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. (Cited on pages 3, 4, 5, 8, 33, 38, 39, 41, 42, 43, 44, 48, 49 and 53.)

[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. (Cited on pages 3, 4, 8 and 42.)

[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. (Cited on page 59.)

[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. (Cited on page 8.)

[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. (Cited on pages 38, 41 and 44.)

[BBR18] Karthikeyan Bhargavan, Richard Barnes, and Eric Rescorla. TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups. May 2018. (Cited on pages 38 and 41.)

[BBS98] 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. (Cited on pages 4 and 59.)

[BDR20] Alexander Bienstock, Yevgeniy Dodis, and Paul Rösler. On the price of concurrency in group ratcheting protocols. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 198–228. Springer, Heidelberg, November 2020. (Cited on page 42.)

[Ben89] Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, 1989. (Cited on pages 8, 17 and 21.)

[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. (Cited on page 54.)

[BHR12] 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. (Cited on page 7.)

[BHY09] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 1–35. Springer, Heidelberg, April 2009. (Cited on page 7.)

[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. (Cited on page 3.)

[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. (Cited on page 8.)

[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. (Cited on page 54.)

[CDG01] Fan Chung, Persi Diaconis, and Ronald Graham. Combinatorics for the east model. *Advances in Applied Mathematics*, 27(1):192–206, 2001. (Cited on pages 9 and 21.)

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *28th ACM STOC*, pages 639–648. ACM Press, May 1996. (Cited on page 7.)

[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. (Cited on page 5.)

[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. (Cited on page 61.)

[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. (Cited on page 7.)

[DKW11] 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. (Cited on page 8.)

[DNRS99] Cynthia Dwork, Moni Naor, Omer Reingold, and Larry J. Stockmeyer. Magic functions. In *40th FOCS*, pages 523–534. IEEE Computer Society Press, October 1999. (Cited on page 7.)

[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. (Cited on page 8.)

[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. (Cited on pages 5 and 7.)

[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. (Cited on pages 4, 5, 8, 20, 59, 60, 61, 62 and 63.)

[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. (Cited on pages 5, 7, 8 and 54.)

[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. (Cited on page 41.)

[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. (Cited on page 8.)

[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. (Cited on page 54.)

[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. (Cited on page 61.)

[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. (Cited on page 40.)

[GOS18] Sanjam Garg, Rafail Ostrovsky, and Akshayaram Srinivasan. Adaptive garbled RAM from laconic oblivious transfer. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 515–544. Springer, Heidelberg, August 2018. (Cited on page 7.)

[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. (Cited on page 7.)

[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. (Cited on page 8.)

[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. (Cited on page 8.)

[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. (Cited on page 7.)

[HKSS14] D. Hefetz, M. Krivelevich, M. Stojakovic, and T Szabó. *Positional Games*. Birkhäuser Basel, 2014. (Cited on page 12.)

[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. (Cited on pages 8 and 15.)

[JKK$^+$17] 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. (Cited on pages 4, 5, 6, 7, 9, 10, 17, 20, 33, 54 and 59.)

[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. (Cited on page 7.)

[KKPW] Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Daniel Wichs. Limits on the adaptive security of Yao's garbling, *CRYPTO 2021*, To Appear. (Cited on pages 6, 8 and 59.)

[KNYY20] Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Compact NIZKs from standard assumptions on bilinear maps. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 379–409. Springer, Heidelberg, May 2020. (Cited on pages 4 and 8.)

[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. (Cited on page 54.)

[Krá01] Richard Královič. *Time and Space Complexity of Reversible Pebbling*, pages 292–303. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. (Cited on page 9.)

[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. (Cited on page 8.)

[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. (Cited on pages 4, 7 and 59.)

[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. (Cited on page 21.)

[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. (Cited on page 8.)

[MP04] Daniele Micciancio and Saurabh Panjwani. Optimal communication complexity of generic multicast key distribution. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 153–170. Springer, Heidelberg, May 2004. (Cited on page 42.)

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 111–126. Springer, Heidelberg, August 2002. (Cited on page 7.)

[Nor15] Jakob Nordström. *New Wine into Old Wineskins: A Survey of SomePebbling Classics with Supplemental Results*. 2015. (Cited on page 9.)

[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. (Cited on pages 3, 4, 5, 7, 10, 33 and 41.)

[Pap85] Christos H. Papadimitriou. Games against nature. *Journal of Computer and System Sciences*, 31(2):288 – 301, 1985. (Cited on page 13.)

[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. (Cited on page 8.)

[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. (Cited on page 8.)

[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. (Cited on pages 8 and 14.)

[Rud88] Steven Rudich. *Limits on the Provable Consequences of One-way Functions*. PhD thesis, EECS Department, University of California, Berkeley, Dec 1988. (Cited on page 8.)

[Sav98] John E. Savage. *Models of computation - exploring the power of computing*. Addison-Wesley, 1998. (Cited on pages 9, 11 and 24.)

[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. (Cited on page 8.)

[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. (Cited on pages 5 and 41.)

[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`. (Cited on page 5.)