The Power of Undirected Rewindings for Adaptive Security

Dennis Hofheinz, Julia Kastner, and Karen Klein

Department of Computer Science
ETH Zurich, Switzerland
{hofheinz,julia.kastner,karen.klein}@inf.ethz.ch

Abstract. Existing proofs of adaptive security (e.g., in settings in which decryption keys are adaptively revealed) often rely on guessing arguments. Such guessing arguments can be simple (and, e.g., just involve guessing which keys are revealed), or more complex "partitioning" arguments. Since guessing directly and negatively impacts the loss of the corresponding security reduction, this leads to black-box lower bounds for a number of cryptographic scenarios that involve adaptive security.
In this work, we provide an alternative to such guessing arguments: instead of guessing in a security reduction which adaptive choices an adversary $\mathcal{A}$ makes, we *rewind* $\mathcal{A}$ many times until we can successfully embed a given computational challenge. The main benefit of using rewindings is that these rewindings can be arranged sequentially, and the corresponding reduction loss only accumulates additively (instead of multiplicatively, as with guessing). The main technical challenge is to show that $\mathcal{A}$'s success is not negatively affected after (potentially many) rewindings. To this end, we develop a machinery for "undirected" rewindings that preserve $\mathcal{A}$'s success across (potentially many) rewindings.
We use this strategy to show
– security of the "Logical Key Hierarchy" protocol underlying the popular TreeKEM key management protocol, and
– security of the Goldreich-Goldwasser-Micali (GGM) pseudorandom function (PRF) as a prefix-constrained PRF.
In both cases, we provide the first polynomial reductions to standard assumptions (i.e., to IND-CPA and PRG security, respectively), and in case of the GGM PRF, we also circumvent an existing lower bound.

1 Introduction

Security reductions. The security of most cryptographic primitives implies P $\neq$ NP, and hence we cannot expect to simply prove them secure. Instead, we typically rely on *security reductions* that transform a given adversary $\mathcal{A}$ on the primitive into a problem solver $\mathcal{S}$ for a given computational problem.[1] For convincing security guarantees, we often desire reductions to very simple and "static" problems like integer factorization or the discrete logarithm problem in

[1]Often, $\mathcal{S}$ itself is also denoted as the reduction.

a given group. On the other hand, certain security notions are complex and may give $\mathcal{A}$ a lot of freedom to adaptively influence what information is available during an attack.

Adaptive security example: signatures. As a concrete example, the standard notion of security for digital signatures, "EUF-CMA security" [GMR88], allows an adversary $\mathcal{A}$ to receive signatures for arbitrarily and adaptively chosen (by $\mathcal{A}$) messages before expecting $\mathcal{A}$ to actually forge a new signature. Hence, if we view a signature scheme as a collection of "signature generation" problem instances (one for each message m), then $\mathcal{A}$ first expects to see many solutions to adaptively chosen instances before generating a solution to a new instance by itself. Now a security reduction must find a way to embed its own computational challenge X^* into this signature setting, in a way such that all instance solutions requested by $\mathcal{A}$ can be solved, but $\mathcal{A}$'s final forgery implies a solution to X^*.

For signatures, several strategies were found to overcome this difficulty. For instance, "partitioning reductions" (explicitly investigated in [Cor02; Wat05; HK08], but implicit in many earlier works) embed a given computational challenge X^* in a certain fraction ρ of all signature generation instances (i.e., messages). The hope is that if we choose ρ right, then with small but significant probability, X^* is embedded only in the message on which $\mathcal{A}$ finally forges, but not in any message that needs to be signed by the reduction. This leads to a successful reduction that however fails with a high probability.

Abstraction: guessing strategies. We might (informally, and only for the purpose of this exposition) call such reduction strategies "guessing strategies". These reductions make certain random guesses about $\mathcal{A}$'s behavior, and fail if those guesses are not accurate. For certain types of primitives, constructions, or reductions it is often possible to show that there are no better strategies (in terms of reduction loss) than guessing strategies:

- For unique [Cor02] or rerandomizable [HJK12] signature schemes, a certain[2] class of reductions must have a loss that is linear in the number of signature queries.
- For the security of secret-key encryption under adaptive corruptions (as formalized by the "generalized selective decryption" notion, which was introduced to prove adaptive security of the "logical key hierarchy" protocol [Pan07; WGL00]), a similar class of "straight-line" reductions must have a *superpolynomial* reduction loss [Kam+21].
- Similar superpolynomial lower bounds [Kam+21] affect the related "TreeKEM" protocol [BBR18] for continuous group key agreement, and the GGM pseudorandom function [GGM84a] when viewed as a prefix-constrained PRF [Kia+13; BW13; BGI14].

More examples of such lower bounds include specific signature schemes [FF13], certain types of encryption schemes [LW14; Bad+16; Kam+21], non-interactive key exchange [Bad+16], and even (composable) zero-knowledge protocols [Can+01].

[2]These reductions must use $\mathcal{A}$ in a black-box way and the corresponding computational problem must be non-interactive. This covers a large class of existing reductions.

The intuition for these lower bounds is that the reduction can be forced to guess many of $\mathcal{A}$'s choices simultaneously and early in the security experiment. This induces a reduction loss that is related to the *product* of the probabilities that each guess (for each of $\mathcal{A}$'s choices) is correct. As a consequence, most (although not all) of the above bounds only consider non-rewinding reductions, i.e., reductions that run an adversary $\mathcal{A}$ in a black-box and straightline manner.[3]

Our contribution. In this work, we consider *rewinding* as an alternative to *guessing* $\mathcal{A}$'s choices. Hence, we design reductions that do not guess, e.g., which parties $\mathcal{A}$ corrupts, but instead (a) run $\mathcal{A}$ in a setting without any embedded challenges, and then (b) rewind and rerun $\mathcal{A}$ in a setting in which challenges have been embedded based on $\mathcal{A}$'s choices in the first run. Of course, it is not clear a priori why this might work (since $\mathcal{A}$'s choices might depend on concrete values that have changed during the runs), and we give details on our concrete strategy and necessary technical conditions below.

The benefit of the use of rewindings over guesses is that guessing n choices of $\mathcal{A}$ simultaneously leads to a reduction loss *exponential* in n. In contrast, with our strategy, the corresponding number of rewindings (when arranged carefully) and thus the reduction loss is only *polynomial* in n.

We apply this idea to two of the above settings. We show

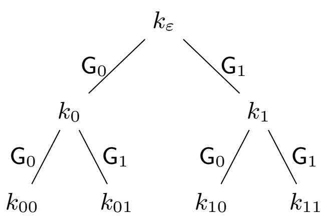
– security of the "logical key hierarchy" (LKH) protocol assuming IND-CPA security of the underlying secret-key encryption scheme, and
– security of the GGM PRF as a prefix-constrained PRF, assuming pseudorandomness of the underlying pseudorandom generator.

The corresponding security reductions have a polynomial loss, and thus circumvent the above-mentioned lower bounds.[4] We believe that these results hint at the potential of using rewindings instead of guessing strategies.

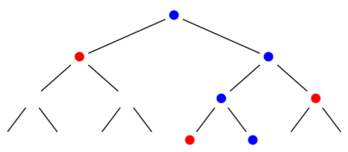
More related work on rewindings. Rewindings have already numerous applications in cryptography, including zero-knowledge simulators [GMW87], signature [PS96] and even encryption schemes [Kuc+20]. In some cases, even complex *nested* rewinding strategies have proved useful [RK99; Can+01]. Jumping ahead, one key difference to our approach is that existing works rewind to a *particular point* in a previous run. For instance, [PS96] analyze a Fiat-Shamir-based [FS87] signature scheme in the random oracle model, and rewind an adversary $\mathcal{A}$ to the point where a particular random oracle query (related to $\mathcal{A}$'s eventual forgery) is made. As we will explain below, this means that the second run (after $\mathcal{A}$'s rewinding) may not have the same distribution as the first run (before the rewinding). In other words, such "directed rewindings" may change $\mathcal{A}$'s success probability, and complex technical tools like the "forking lemma" [PS96] may be necessary to analyze such scenarios.

[3]An interesting exception is the work of [Cor02] that does consider rewinding reductions. This is possible because the corresponding signature setting and adversary $\mathcal{A}$ is particularly simple (so that rewinding $\mathcal{A}$ is of little use).

[4]Strictly speaking, in case of LKH, lower bounds are only known for the (very related) "TreeKEM" protocol [Kam+21].



(a) The GGM PRF evaluation.



(b) GGM PRF tree with randomized keys along the path (blue ●) and co-path (red ●) to k_{101}, as desirable when $x^* = 101$ is selected as challenge.

Fig. 1: The GGM PRF evaluation (Fig. 1a) and tree (Fig. 1b).

As a consequence, "directed rewindings" may not scale well to settings in which we want to rewind very liberally to replace complex guessing strategies. In contrast, we develop a very different, "undirected" rewinding machinery that will preserve $\mathcal{A}$'s output distribution across rewindings.

1.1 Technical overview

Example: the GGM PRF. Our approach is perhaps easiest to showcase with the pseudorandom function of Goldreich, Goldwasser, and Micali [GGM84a] (henceforth "GGM PRF"). Recall that the GGM PRF starts from a length-doubling pseudorandom generator $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ written as $\mathsf{G}(s) = \mathsf{G}_0(s) \| \mathsf{G}_1(s)$ for $\mathsf{G}_0, \mathsf{G}_1 : \{0,1\}^\lambda \to \{0,1\}^\lambda$. The PRF output $\mathsf{F}(k, x^*)$, for a PRF key $k \in \{0,1\}^\lambda$ and an input $x^* \in \{0,1\}^d$ (for some fixed input length d) is defined as k_{x^*}, which is iteratively given through

$$k_\varepsilon := k, \quad \forall x \in \{0,1\}^{<d}, b \in \{0,1\} : \; k_{x\|b} := \mathsf{G}_b(k_x). \tag{1}$$

This evaluation process is illustrated in Fig. 1a.

The GGM PRF as a prefix-constrained PRF. The GGM PRF is known to have a very nice key delegation feature [Kia+13; BW13; BGI14]. Namely, observe that an "intermediate key" $k_{x'}$ (for $0 < |x'| < d$) as defined in (1) allows to compute precisely those outputs k_x that start with x'. Furthermore, $k_{x'}$ can be efficiently computed from k_ε, or even any intermediate key $k_{x''}$ for an x'' that is a prefix of x'. Of course, for security we would also hope that $k_{x'}$ does not reveal anything about outputs k_{x^*} for x^* that do *not* start with x'.

This type of delegation property is called "prefix-constrainability". The corresponding security experiment requires pseudorandomness of a single output $\mathsf{F}(k, x^*) = k_{x^*}$ for an adversarially chosen x^*, *even* when that adversary has adaptive access to many constrained keys $k_{x'}$ (where of course no x' may be a prefix of x^*).

Selective security. To set the stage, we first observe that it is relatively easy to prove a *selective* version of prefix-constrainability, in which the adversary $\mathcal{A}$ has to commit in advance to x^*. Namely, if x^* is known, a reduction could proceed in a hybrid argument and successively embed G-challenges along the path from k_ε

to k_{x^*}. This embedding is possible, since $\mathcal{A}$ may not ask for constrained keys $k_{x'}$ that are ancestors of k_{x^*}, and thus no G-preimage will ever have to be revealed. Furthermore, this way, gradually all keys on the path (and co-path) to k_{x^*} will be randomized (see Fig. 1b). Once k_{x^*} itself is independently random, $\mathcal{A}$ cannot win the security game anymore.

The difficulty. Interestingly, the situation is completely different in the adaptive setting, when $\mathcal{A}$ may ask for constrained keys $k_{x'}$ *before* committing to x^*. The difficulty lies in the fact that $\mathcal{A}$ can force a security reduction to "commit" to a large part of the evaluation tree from Fig. 1a by asking for constrained keys $k_{x'}$, but without finally committing itself to the challenge x^*. This forces a reduction (to the pseudorandomness of G) to decide early on where challenge G-images are to be embedded, and in essence guess (parts of) x^* in advance.

Viewed from a different angle, trying to proceed as in the selective case will require to randomize k_{x^*} and thus, at least in parts, intermediate keys $k_{x'}$ for prefixes x' of x^*, all while being able to (for constrained key queries) explain the rest of the evaluation tree as being pseudorandom. Since $\mathcal{A}$ may choose x^* very late, however, it is not at all clear how to suitably embed G-challenges for this randomization.

In fact, [Kam+21] formally prove that no black-box, non-rewinding reduction with polynomial security loss exists in this setting for the GGM PRF. (The hardness of achieving adaptive prefix-constrainability even with more powerful tools and more complex PRF constructions is also explicitly mentioned in [Dav+20].) Notwithstanding, [Fuc+14] do manage to give a black-box, non-rewinding reduction for the GGM PRF with only slightly superpolynomial loss. Their argument is a clever "pebbling strategy" that manages to, informally, guess x^* not all at once, but only in parts. Still, even with this clever strategy, there will be times when several parts of x^* have to be guessed simultaneously, which leads to a superpolynomial security loss.

Our solution. Our approach is not to guess x^* at all, but to *rewind* an adversary $\mathcal{A}$ in the above setting many times, embedding more and more G-values along the challenge path to k_{x^*} (as in Fig. 1b). Of course, when changing $\mathcal{A}$'s view in any way (and in particular when embedding G-values), the challenge preimage x^* chosen by $\mathcal{A}$ may change completely. Thus, we will have to ensure that during those rewindings, already embedded G-values remain on that challenge path.

To make things clearer, let us first describe a strategy that only *almost* works, but demonstrates the basic ideas. Concretely, consider an adversary $\mathcal{A}$ attacking the GGM PRF of depth $d = 3$, as in Fig. 1b. Without loss of generality, we may assume that $\mathcal{A}$ eventually asks for all keys $k_{x'}$ for x' on the co-path of the challenge path to x^*. (That is, if $\mathcal{A}$ eventually nominates $x^* = 101$ as challenge, we assume that $\mathcal{A}$ will have asked for $k_{x'}$ for all $x' \in \{0, 11, 100\}$.) Intuitively, this means that $\mathcal{A}$ eventually knows the whole evaluation tree except for the challenge path.

Our (preliminary) reduction proceeds as follows:

1. First, run $\mathcal{A}$ on an evaluation tree with fully known keys k_x (for all $x \in \{0,1\}^{\leq d}$). For concreteness, let us say that the challenge input that $\mathcal{A}$ eventually chooses is $x^* = 101$, as in Fig. 1b.
2. Next, rewind $\mathcal{A}$ back to the point in time t_1 at that k_1, the first intermediate key on the challenge path to k_{101}, is computed. (Since *every* constrained key query requires to commit to k_ε, and hence also compute k_1, this means we rewind to the first constrained key query.) Then, rerun $\mathcal{A}$ from this point t_1 onwards with a fresh G-challenge embedded for (the image of) k_ε.
3. Continue with the rewound run that includes an embedded G-challenge for (the image of) k_ε, and may now feature a new $x^* = 001 \neq 101$. Rewind to the point t_2 in time when the second key $k_{x'}$ along the new challenge path, i.e. a key for the length-2 prefix x' of the new x^*, is computed. Rerun $\mathcal{A}$ from this point t_2 onwards with a fresh G-challenge embedded for (the image of) k_0. This embedding is possible since k_0 is already uniformly random (as it is the output of the first embedded G-challenge), and since this is the first query in which k_0 is used. Unfortunately, this embedding also requires that at no point, any ancestor of $k_{x'}$ will have to be revealed, even when later constrained key queries and even x^* may change after the rewinding. Hence, rewind repeatedly[5], until that particular query at time t_2 is the first one to explicitly use the key for the length-1 prefix of (the now potentially different) x^*.[6] Note that by definition, $t_2 > t_1$, so the new rewindings will not replace the previously embedded G-challenge.
4. Continue with this process for longer prefixes of x^*, eventually embedding the output of a G-challenge into $\mathcal{A}$'s own challenge k_{x^*}. In the resulting run, G-challenges are embedded exactly along the evaluation tree path to the PRF challenge input x^*, as depicted in Fig. 1b for $x^* = 101$. Hence, $\mathcal{A}$'s final output in this run can be used to break the pseudorandomness of G.

We remark that the choice of queries that involve *prefixes* of x^* resembles similar techniques in the signature setting [NY89; HW09; BK10] (and has also been used as an ingredient in the context of the GGM PRF [Fuc+14]).

A technical complication. . . While the previous description is largely accurate, it glosses over one crucial detail. Namely, recall that we make liberal use of rewindings. Moreover, our final argument implicitly uses that runs generated through rewindings have (at least computationally) the same distribution as non-rewound ones. In particular, $\mathcal{A}$'s success probability must be preserved across rewindings. But this is not guaranteed with "directed rewindings" as above, where the choice of the point in time to rewind to is chosen adaptively, based on what happened pre-rewinding.

[5]Polynomially many rewindings will suffice (with high probability), since the condition we require to be preserved is not overly specific.

[6]We are simplifying here. In particular, this step assumes that k_1 is already random, not only a G-challenge. Our actual proof uses a hybrid argument, much like the one for selective security from above.

To explain the issue, consider the toy example of a one-dimensional random walk

$$T := \sum_{i=1}^{n} t_i \quad \text{for independently uniformly random } t_i \in \{-1, 1\}$$

of length n. Clearly, the expected value of T is 0. On the other hand, if we

1. sample T (and all t_i) as above,
2. then fix the smallest index $m \in \{1, \ldots, n-1\}$ of a local maximum (such that $t_m = 1$ and $t_{m+1} = -1$)[7] and
3. resample T conditioned on $(t_1, \ldots, t_m)$ (i.e., keeping the values of $(t_1, \ldots, t_m)$),

then the resulting T has a positive expected value.[8] A similar situation may arise in our reduction above: we resample runs with $\mathcal{A}$ conditioned on run prefixes, where the prefix length is based on $\mathcal{A}$'s behavior up to that point. We cannot guarantee that this resampling does not bias, e.g., $\mathcal{A}$'s success probability.

. . . resolved. We will overcome this obstacle with "undirected" rewindings, that rewind to a pre-determined point in time, independently of what happened in the run prior to rewinding. Going back to the toy example of a random walk T, observe that when sampling T and then conditioning on $(t_1, \ldots, t_m)$ for any a-priori fixed prefix length m does not change T's distribution. (Since all t_i are independently random, this is just a complicated way of sampling a single T.) More generally, we will show that rewindings as in our reduction do not change run distributions if the condition itself that we seek to be preserved during rewindings does not depend on the initial run.

Recall that for us, the conditions to be preserved across rewindings are of the form "the query at time t_1 is the first one to explicitly use $k_{x'}$ for a prefix x' of x^*". The problem with this formulation is that this time t_1 depends on the previously sampled run. Our actual solution is hence a bit more complex: we rewind (repeatedly) at *every* time index t, and preserve a function on runs across rewindings. This function is of the form "output the length of the longest prefix x' of x^* such that a key $k_{x'}$ was explicitly computed/defined before or at time t". Preserving this function value across rewindings allows to implement the above reduction strategy, although at the cost of a higher (but still polynomial) number of rewindings.

Second application: encryption security under adaptive corruptions. Our second application concerns the "Logical Key Hierarchy" (LKH) protocol [WGL00] (in the "fixed version" [Pan07]) and the related "TreeKEM" protocol [BBR18] for continuous group key agreement. In these protocols, a binary

[7]Such an m exists except with probability $(n+1)/2^n$. Hence, we ignore the case that no such m exists.

[8]For any fixed local maximum m, there are only m possibilities for the values of $(t_1, \ldots, t_{m-1})$: those with $(t_1, \ldots, t_{m-1}) = (-1, \ldots, -1, 1, \ldots, 1)$, all equally likely. Since $t_m = 1$, this means that the expected value of $\sum_{i=1}^{m} t_i$ is 1 (conditioned on the event that m exists). Also, by our resampling strategy, $\sum_{i=m+1}^{n} t_i$ has expected value 0. Since m exists with high probability, thus $\sum_{i=1}^{m} t_i$ has positive expectation.

tree of decryption keys k_x (for $x \in \{0,1\}^{\leq d}$) is arranged as in Fig. 1a. Unlike with the GGM PRF setting, the keys k_x themselves are independently chosen. However, for every $x \in \{0,1\}^{<d}$ and $b \in \{0,1\}$, a ciphertext $c_{x\|b}$ that encrypts k_x under key $k_{x\|b}$ is publicly available.[9]

This setup enables the owner of any leaf key k_x to compute the root key k_ε, which can then be used for group communication. We also consider a dynamic setting, in which users leave or join this group. When user x (i.e., the user who owns k_x) leaves, the shared key k_ε and all keys on the path to k_x are refreshed, along with all ciphertexts that encrypt these keys. This requires an update of only $O(|x|)$ many ciphertexts and keys. Similarly, a join only requires the generation of $O(|x|)$ new ciphertexts.

For security, we desire that an adversary who may adaptively initiate leaves and joins, and who learns the corresponding keys k_x of leaving users, cannot distinguish the eventual refreshed root key k_ε from a random key. We will seek to prove security based on the semantic (or IND-CPA) security of the underlying encryption scheme.

Our overall strategy will be similar to the GGM PRF case. However, we will also need to embed challenge ciphertexts (and not only challenge keys) into runs. Besides, one key difference is that in the LKH security experiment, there is no single challenge leaf x^* (as with the GGM PRF). Instead, we will build upon the intricate "pebbling" strategy of [Jaf+17] to randomize k_ε, only with guesses replaced by rewindings. This will translate into a more complex property to be preserved across rewindings, which also causes a more complex runtime analysis. Concretely, we will have to switch between games with a bounded number of rewindings (to be able to use a reduction to a computational assumption), and ones without such a bound (to be able to switch equivalent preserved properties).

1.2 Roadmap

We recall some relevant preliminaries about probability theory and cryptographic primitives in Section 2. In Section 3, we give an abstract and application-independent version of our rewinding analysis. In the following sections, we consider the GGM PRF (in Section 4) and adaptive encryption security (in Section 5) applications.

2 Preliminaries

2.1 Notation

Security parameter. Throughout the paper, $\lambda \in \mathbb{N}$ denotes the security parameter. Many other variables (such as parameters or distributions) may depend

[9]The LKH and TreeKEM protocols are very similar, with one key difference being that the former uses secret-key encryption, while latter employs public-key encryption. Our results are formulated in the secret-key setting and thus directly apply only to LKH (although we are confident that our strategy can also be used for TreeKEM).

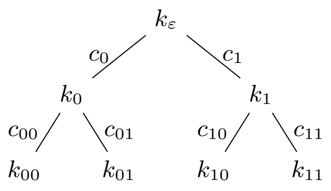


Fig. 2: A depth-2 binary tree with node and edge names.

on λ. A function $\delta = \delta(\lambda)$ is *negligible* (in λ) if $\forall c \in \mathbb{N}\ \exists \lambda_0\ \forall \lambda > \lambda_0 :\ |\delta(\lambda)| < 1/\lambda^c$.

Sets and bitstrings. We write $[n] := \{1, \ldots, n\}$ and $[n]_0 := \{0, 1, \ldots, n\}$. For two sets $\mathcal{X}$, $\mathcal{Y}$ we denote the symmetric difference between $\mathcal{X}$ and $\mathcal{Y}$ as $\mathcal{X} \Delta \mathcal{Y} := (\mathcal{X} \setminus \mathcal{Y}) \cup (\mathcal{Y} \setminus \mathcal{X})$. With $\{0,1\}^n$, $\{0,1\}^{\leq n}$, and $\{0,1\}^{<n}$, we mean all bitstrings of length exactly n, at most n, and less than n, respectively. The lexicographic ordering upon bitstrings x is denoted with $\leq_{\mathsf{lex}}$. If x is a prefix of x', we write $x \leq_{\mathsf{pfx}} x'$, for a proper pefix we write $x <_{\mathsf{pfx}} x'$. For a finite vector $x = (x_1, \ldots, x_n) \in \Sigma^n$ over an alphabet Σ, we denote by $\mathsf{pfx}_j(x)$ the prefix $(x_1, \ldots, x_j)$ of x. The symbol $\|$ denotes string or sequence concatenation.

Tree notation. For our applications, we will consider complete binary trees whose depth we generally denote by d. We derive generic names for nodes and edges from our applications: concretely, we denote the root node as k_ε (where ε is the empty bitstring), and the two child nodes of each node k_x as $k_{x\|0}$ and $k_{x\|1}$. For each $x \in \{0,1\}^{<d}$ and $b \in \{0,1\}$, there is an edge $c_{x\|b}$ between k_x and $k_{x\|b}$. (See Fig. 2 for an example with $d = 2$.) For a binary tree of depth d and a path $P = (k_x, \ldots, k_\varepsilon)$ from a leaf $x \in \{0,1\}^d$ to the root, the *co-path* of P consists of the sibling vertices of the vertices on P. More formally, writing $x = (x_1, \ldots, x_d)$, the co-path consists of the vertices $(k_{\mathsf{pfx}_{d-1}(x)\|(1-x_d)}, k_{\mathsf{pfx}_{d-2}(x)\|(1-x_{d-1})}, \ldots, k_{1-x_1})$.

Probabilities, distributions, and predicates. If $\mathcal{D}$ is a distribution over some set $\mathcal{X}$, then

$$\rho_{\mathcal{D}}(x) := \Pr_{X \leftarrow \mathcal{D}}[X = x].$$

Furthermore, if $f : \mathcal{X} \to \mathcal{Y}$ is a function, then $f(\mathcal{D})$ denotes the distribution over $\mathcal{Y}$ that arises by applying f to values sampled from $\mathcal{D}$. If $\mathrm{P} : \mathcal{X} \to \{\mathtt{true}, \mathtt{false}\}$ is a predicate, then $\mathcal{D} \mid \mathrm{P}$ denotes the conditional distribution of $\mathcal{D}$ conditioned on $\mathrm{P}(\cdot) = \mathtt{true}$. As a special case, we consider equalities as predicates P in the above sense, and may write, e.g., $\mathcal{D} \mid [f(\cdot) = y]$.

For two random variables X, Y (which may depend on the security parameter λ), we write $X \equiv Y$ if they are identically distributed, $X \overset{\mathrm{s}}{\approx}_\delta Y$ if their statistical distance is at most δ, and $X \overset{\mathrm{c}}{\approx} Y$ if they are computationally indistinguishable.

2.2 Probability theory

We will need a special Chernoff bound. We state without proof:

Lemma 1. *Let $E_1, \dots, E_\ell$ be independent events that each occur with probability p. Then*

$$\Pr\Big[\bigvee_{t=1}^{\ell} E_t\Big] \;\geq\; 1 - 1/e^{\ell p/2}.$$

The following lemma is straightforward:

Lemma 2. *Let $\mathcal{D}$ be a distribution over $\mathcal{X}$, and $f : \mathcal{X} \to \mathcal{Y}$ be a function. Consider random variables X, X_0 with*

$$X_0 \leftarrow \mathcal{D} \qquad\qquad X \leftarrow \mathcal{D} \mid [f(\cdot) = f(X_0)].$$

Then X is distributed according to $\mathcal{D}$, i.e., we have $\forall x \in \mathcal{X} : \Pr[X = x] = \Pr[X_0 = x] = \rho_{\mathcal{D}}(x)$.

Intuitively, Lemma 2 states that resampling conditioned on a "current value" $f(X_0)$ does not change the distribution.

Proof.

$$\begin{aligned}
\Pr[X = x] &= \sum_{y \in \mathcal{Y}} \Pr[X = x \wedge f(X) = y] \\
&= \sum_{y \in \mathcal{Y}} \Pr[X = x \mid f(X) = y] \cdot \Pr[f(X) = y] \\
&= \sum_{y \in \mathcal{Y}} \Pr[X_0 = x \mid f(X_0) = y] \cdot \Pr[f(X_0) = y] \\
&= \sum_{y \in \mathcal{Y}} \Pr[X_0 = x \wedge f(X_0) = y] \;=\; \Pr[X_0 = x].
\end{aligned}$$

□

The next lemma is a probabilistic version of the "bucket lemma" of Kastner, Loss, and Xu [KLX22], which in turn generalizes the "forking lemma" of Pointcheval and Stern [PS96].

Lemma 3. *Let $\mathcal{D}$ be a distribution over $\mathcal{X}$, and $f : \mathcal{X} \to \mathcal{Y}$ be a function with finite range $\mathcal{Y}$. For any $\alpha \in [0, 1]$,*

$$\Pr_{X \leftarrow \mathcal{D}}\big[\, \rho_{f(\mathcal{D})}(f(X)) \geq \alpha \,\big] \;\geq\; 1 - \alpha \cdot |\mathcal{Y}|.$$

Intuitively, Lemma 3 states that it is likely that an $X \leftarrow \mathcal{D}$ has a "somewhat common" value of $f(X)$.

Proof.

$$\begin{aligned}
\Pr_{X \leftarrow \mathcal{D}}\big[\, \rho_{f(\mathcal{D})}(f(X)) \geq \alpha \,\big] &= \sum_{\substack{y \in \mathcal{Y} \\ \rho_{f(\mathcal{D})}(y) \geq \alpha}} \rho_{f(\mathcal{D})}(y) \\
&= \sum_{y \in \mathcal{Y}} \rho_{f(\mathcal{D})}(y) - \sum_{\substack{y \in \mathcal{Y} \\ \rho_{f(\mathcal{D})}(y) < \alpha}} \rho_{f(\mathcal{D})}(y) \;\geq\; 1 - |\mathcal{Y}| \cdot \alpha.
\end{aligned}$$

□

2.3 Cryptographic primitives

For convenience, we define pseudorandom number generators (PRGs) with a multi-instance security notion (that is however easily seen to be polynomially equivalent to the ordinary one-instance notion using a hybrid argument):

Definition 1 ($(\mathcal{Q}, t, \delta)$-hard pseudorandom generator (PRG)). *An efficiently computable function $\mathsf{G} : \{0,1\}^n \mapsto \{0,1\}^m$ with $m > n$ is a $(\mathcal{Q}, t, \delta)$-hard* pseudo-random generator (PRG) *if every probabilistic adversary $\mathcal{A}$ that makes at most $\mathcal{Q}$ oracle queries and runs in time at most t satisfies $|\mathrm{Adv}^{\mathsf{PR}}_{\mathsf{G},\mathcal{A}}(\lambda)| \leq \delta$, where*

$$\mathrm{Adv}^{\mathsf{PR}}_{\mathsf{G},\mathcal{A}}(\lambda) := \Pr[\mathsf{MI\text{-}PRG}^{\mathcal{A}}_{\mathsf{G}}(\lambda) = 1] - 1/2$$

for the experiment $\mathsf{MI\text{-}PRG}^{\mathcal{A}}_{\mathsf{G}}$ defined in Fig. 3.

Asymptotically, we say that G is a secure PRG if for all polynomials $\mathcal{Q}, t$ in λ, there is a negligible $\delta = \delta(\lambda)$, so that G is a $(\mathcal{Q}, t, \delta)$-hard PRG.

Algorithm 1: $\mathsf{MI\text{-}PRG}^{\mathcal{A}}_{\mathsf{F}}(\lambda)$

```
1 $b \leftarrow \{0,1\}$
2 $b' \leftarrow \mathcal{A}^{\text{challenge}}(1^\lambda)$
3 return $[b = b']$
```

Algorithm 2: challenge()

```
1 $s \leftarrow \{0,1\}^n$
2 $y_0 := \mathsf{G}(s)$; $y_1 \leftarrow \{0,1\}^m$
3 return $y_b$
```

Fig. 3: Multi-instance PRG indistinguishability game

In our setting, we will only be interested in PRGs with $n = \lambda$ and $m = 2\lambda$.

Definition 2 (Prefix-constrained pseudorandom functions). *Consider an efficiently computable function $\mathsf{F} : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^m$ that takes as input a key $k \in \{0,1\}^\lambda$ and an input $x \in \{0,1\}^n$, and outputs an image $y \in \{0,1\}^m$.*

We say that F is a prefix-constrained pseudorandom function (PC-PRF) *if there are polynomial-time algorithms* constrain *and* ceval *with the following properties:* constrain *may be probabilistic, takes as input a key $k \in \{0,1\}^\lambda$ and a prefix $x' \in \{0,1\}^{\leq n}$, and outputs a constrained key $k_{x'}$.* ceval *is deterministic, takes as input such a constrained key $k_{x'}$ and an input $x \in \{0,1\}^n$, and outputs an image $y \in \{0,1\}^m$. We require that for all λ, $k \in \{0,1\}^\lambda$, $k_{x'} \leftarrow \mathsf{constrain}(k, x')$, and $x \in \{0,1\}^n$ with $x' \leq_{\mathsf{pfx}} x$, we have*

$$\mathsf{ceval}(k_{x'}, x) \; = \; \mathsf{F}(k, x).$$

The main security property of PC-PRFs is indistinguishability:

Definition 3 ($(\mathcal{Q}, t, \delta)$-indistinguishability for PC-PRFs). *Let F be a PC-PRF as in Definition 2. We say that F is $(\mathcal{Q}, t, \delta)$*-indistinguishable *if for every probabilistic adversary $\mathcal{A}$ that runs in time at most t, makes at most $\mathcal{Q}$ queries* *to the* constrain *oracle and at most one query to the* challenge *oracle in the* $\mathsf{PC\text{-}PRF}_{\mathsf{F},\mathcal{A}}$ *experiment, we have* $|\mathrm{Adv}^{\mathsf{PC\text{-}PRF}}_{\mathsf{F},\mathcal{A}}(\lambda)| \le \delta$*, where*

$$\mathrm{Adv}^{\mathsf{PC\text{-}PRF}}_{\mathsf{F},\mathcal{A}}(\lambda) := \Pr[\mathsf{PC\text{-}PRF}^{\mathcal{A}}_{\mathsf{F}}(\lambda) = 1] - 1/2$$

for the experiment $\mathsf{PC\text{-}PRF}^{\mathcal{A}}_{\mathsf{F}}$ *defined in Fig. 4.*

Asymptotically, we say that F *is an indistinguishable PC-PRF if for all polynomials* $\mathcal{Q}, t$ *in* λ*, there is a negligible* $\delta = \delta(\lambda)$*, so that* F *is* $(\mathcal{Q}, t, \delta)$*-indistinguishable.*

```
Algorithm 3: $\mathsf{PC\text{-}PRF}^{\mathcal{A}}_{\mathsf{F}}(\lambda)$
1 $b \leftarrow \{0,1\}$
2 $k \leftarrow \{0,1\}^{\lambda}$
3 $X := \emptyset$
4 $x^* := \varepsilon$
5 $b' \leftarrow \mathcal{A}^{\text{constrain},\text{challenge}}(1^{\lambda})$
6 return $[b = b']$
```

```
Algorithm 4: constrain($x'$)
1 if $x' \le_{\mathsf{pfx}} x^*$ then return $\perp$
2 $X := X \cup \{x'\}$
3 $k_{x'} \leftarrow \mathsf{constrain}(k, x')$
4 return $k_{x'}$
```

```
Algorithm 5: challenge($x$)
1 if $\exists x' \in X : x' \le_{\mathsf{pfx}} x$ then return $\perp$
2 $x^* := x$
3 $y_0^* := \mathsf{F}(k, x)$; $y_1^* \leftarrow \{0,1\}^m$
4 return $y_b^*$
```

Fig. 4: CP-PRF indistinguishability game

Definition 4 (Secret-key encryption). *A* secret-key encryption scheme *consists of the following algorithms* $\mathsf{SKE} = (\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec})$*:*

$\mathbf{Gen}(1^{\lambda})$ *takes as input the security parameter encoded in unary, and outputs a key* k*.*

$\mathbf{Enc}(k, m)$ *takes as input a key* k *and a message* $m \in \mathcal{M}$*, and outputs a ciphertext* c*.*

$\mathbf{Dec}(k, c)$ *takes as input a key* k *and a ciphertext* c *and outputs either a message* $m \in \mathcal{M}$ *or an error symbol* $\perp$*.*

We require correctness*, i.e.,* $\forall \lambda$*, and* $m \in \mathcal{M}$*, we have*

$$\Pr[\mathbf{Dec}(k, c) = m \mid k \leftarrow \mathbf{Gen}(1^{\lambda}), c \leftarrow \mathbf{Enc}(k, m)] = 1.$$

Definition 5 (Many-user, many-ciphertext SKE indistinguishability). *A secret-key encryption scheme* SKE *is* $(\mathcal{Q}_{\mathsf{LoR}}, \mathcal{Q}_{\mathsf{NU}}, t, \delta)$*-indistinguishable under chosen-plaintext attacks (short:* $(\mathcal{Q}_{\mathsf{ctxt}}, \mathcal{Q}_{\mathsf{NU}}, t, \delta)$-$\mathsf{IND\text{-}CPA}$ *secure) if every probabilistic adversary* $\mathcal{A}$ *that runs in time at most* t*, and makes at most* $\mathcal{Q}_{\mathsf{LoR}}$ *and* $\mathcal{Q}_{\mathsf{NU}}$ *queries to the LoR and NU oracles below, respectively, satisfies* $|\mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE},\mathcal{A}}(\lambda)| \le \delta$*, where*

$$\mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE},\mathcal{A}}(\lambda) := \Pr[\mathsf{IND\text{-}CPA}^{\mathcal{A}}_{\mathsf{SKE}}(\lambda) = 1] - 1/2$$

Algorithm 6: $\mathsf{IND\text{-}CPA}^{\mathcal{A}}_{\mathsf{SKE}}(\lambda)$

```
1 $b \leftarrow \{0,1\}$
2 $U := [\,]$    // empty array
3 $b' \leftarrow \mathcal{A}^{LoR,NU}(1^\lambda)$
4 return $[b = b']$
```

Algorithm 7: NU

```
1 $k \leftarrow \mathbf{Gen}(1^\lambda)$
2 $U[\mathsf{len}(U)+1] := k$    // append to array
```

Algorithm 8: $LoR(i, m_0, m_1)$

```
1 $c \leftarrow \mathbf{Enc}(U[i], m_b)$  // $\perp$ if $U[i]$ undef'd
2 return $c$
```

Fig. 5: Many-user, many-challenge IND-CPA game

for the $\mathsf{IND\text{-}CPA}^{\mathcal{A}}_{\mathsf{SKE}}$ *experiment defined in Fig. 5.*

Asymptotically, we say that SKE *is* $\mathsf{IND\text{-}CPA}$ *secure if for all polynomials* $\mathcal{Q}_{\mathsf{LoR}}, \mathcal{Q}_{\mathsf{NU}}, t$ *in* λ*, there is a negligible* $\delta = \delta(\lambda)$*, so that* SKE *is* $(\mathcal{Q}_{\mathsf{LoR}}, \mathcal{Q}_{\mathsf{NU}}, t, \delta)$-$\mathsf{IND\text{-}CPA}$ *secure.*

We remark that this many-user, many-ciphertext formulation of IND-CPA security is polynomially equivalent (using a standard hybrid argument) to the traditional one-user, one-ciphertext formulation (as in, e.g., [Bel+97]).

3 Analysis of a repeated resampling algorithm

Overview. In this section, we will provide a few helper results for our upcoming applications. Specifically, we will investigate what happens when we first sample some X_0 from a distribution (which can be a run with an adversary $\mathcal{A}$), and then resample conditioned on parts of X_0. (This latter operation corresponds to rewinding and rerunning $\mathcal{A}$ until a certain property of the full run is preserved.)

As explained in the introduction, the main difference to previous rewinding treatments is that we consider “undirected” rewindings, which translates to resampling conditioned on a-priori fixed properties of X_0. This will enable us to deduce that this resampling does not change the output distribution, and that resampling is likely to preserve any “sufficiently common” property of the initial X_0 in the process.

Generic framework. In the following, let $\mathcal{D}$ be a distribution over some set $\mathcal{X}$, and assume functions $f_1, \ldots, f_{\mathcal{T}} : \mathcal{X} \to \mathcal{Y}$ for a finite set $\mathcal{Y}$. Now consider Algorithm 9. Algorithm 9 starts with a fresh $\mathcal{D}$-sample, and then repeatedly resamples while preserving the value of the functions f_t on those samples. We have:

Lemma 4. *All* X_t *defined through Algorithm 9 are distributed according to* $\mathcal{D}$*, i.e.,* $\forall t, x : \Pr[X_t = x] = \rho_{\mathcal{D}}(x)$*.*

Proof. For X_0, this is clear. For $X_{t-1} \leftarrow \mathcal{D}$, we obtain $\forall x : \Pr[X_t = x] = \rho_{\mathcal{D}}(x)$ by applying Lemma 2. □

Algorithm 9: Repeated resampling, generic

```
Input: $\mathcal{D}, f_1, \ldots, f_{\mathcal{T}}$
1 $X_0 \leftarrow \mathcal{D}$
2 for $t := 1$ to $\mathcal{T}$ do
3     $X_t \leftarrow \mathcal{D} \mid [f_t(\cdot) = f_t(X_{t-1})]$
4 end
5 return $X_{\mathcal{T}}$
```

This in particular holds for Algorithm 9's output $X_{\mathcal{T}}$. Hence, Algorithm 9 would seem like an unnecessarily complicated way to sample from $\mathcal{D}$. However, in the following, we will refine Algorithm 9 to better capture our upcoming rewinding process.

Algorithm 10: Repeated resampling, split

```
Input: $\mathcal{D}, g_1, \ldots, g_{\mathcal{T}}, h_1, \ldots, h_{\mathcal{T}}$
1 $X_0 \leftarrow \mathcal{D}$
2 for $t := 1$ to $\mathcal{T}$ do
3     repeat
4         $X_t \leftarrow \mathcal{D} \mid [h_t(\cdot) = h_t(X_{t-1})]$
5     until $g_t(X_t) = g_t(X_{t-1})$
6 end
7 return $X_{\mathcal{T}}$
```

Split resampling. Now Algorithm 10 performs the generation of the X_t through a different, yet conceptually equivalent form of resampling. More concretely, Algorithm 10 conditions not only on one function value $f_t(X_{t-1})$, but on two function values $g_t(X_{t-1})$ and $h_t(X_{t-1})$. Here, we assume functions $g_t : \mathcal{X} \to \mathcal{Y}$ and $h_t : \mathcal{X} \to \mathcal{Z}$ for a finite set $\mathcal{Y}$ and a set $\mathcal{Z}$. This "double resampling" is done in a somewhat peculiar way: the distribution already conditioned on $h_t(X_{t-1})$ is sampled until a value X_t with $g_t(X_t) = g_t(X_{t-1})$ appears. Still, we obtain as before:

Lemma 5. *All X_t defined through Algorithm 10 are distributed according to $\mathcal{D}$, i.e., $\forall t, x : \Pr[X_t = x] = \rho_{\mathcal{D}}(x)$.*

Proof. For any $t \in [\mathcal{T}]$, the **repeat** loop samples X_t from

$$\big(\mathcal{D} \mid [h_t(\cdot) = h_t(X_{t-1})]\big) \;\mid\; [g_t(\cdot) = g_t(X_{t-1})] \;=\; \mathcal{D} \mid [f_t(\cdot) = f_t(X_{t-1})]$$

for the function $f_t(X) = (g_t(X), h_t(X))$. Hence, Algorithm 10 is equivalent to Algorithm 9 (for these f_t), and Lemma 4 yields the statement. □

Additionally, we can bound the runtime of Algorithm 10:

Lemma 6. *Let T_t^{rep} be the number of all iterations of the* **repeat** *loop for this value of t in Algorithm 10. For any $\gamma \in (0,1]$, we have*

$$\Pr[\, \forall t \in [\mathcal{T}] : T_t^{\mathsf{rep}} \leq \underbrace{2\mathcal{T} \cdot \ln(2\mathcal{T}/\gamma) \cdot |\mathcal{Y}|/\gamma}_{=:T_{\mathbf{max}}(\mathcal{T},|\mathcal{Y}|,\gamma)} \,] \;\geq\; 1-\gamma, \tag{2}$$

where $\mathcal{Y}$ is the (finite) domain of the g_t.

Proof. First fix a $t \in [\mathcal{T}]$. By Lemma 5, X_{t-1} is distributed according to $\mathcal{D}$. Hence, using Lemma 2, X_{t-1} is also distributed according to

$$\mathcal{D}' \;:=\; \mathcal{D} \,|\, [h_t(\cdot) = h_t(X^*)]$$

for some independently chosen $X^* \leftarrow \mathcal{D}$. Since $h_t(X_t) = h_t(X^*)$ by definition, in each iteration of Line 4, X_t is also distributed according to $\mathcal{D}'$. Now invoke Lemma 3 with $\alpha := \gamma/(2\mathcal{T} \cdot |\mathcal{Y}|)$, distribution $\mathcal{D}'$, and function g_t. This yields

$$\Pr_{X_t \leftarrow \mathcal{D}'}[g_t(X_t) = g_t(X_{t-1})] \;\geq\; \alpha, \tag{3}$$

except with probability $\gamma/(2\mathcal{T})$ (over X_{t-1}).

Recall that T_t^{rep} is the number of iterations of the **repeat** loop for this t. Conditioned on (3), Lemma 1 (instantiated with E_t as the event that the t-th iteration succeeds, $p := \alpha$, and $\ell := 2\ln(2\mathcal{T}/\gamma)/\alpha$) shows

$$\Pr\left[T_t^{\mathsf{rep}} \;\leq\; \frac{2\ln(2\mathcal{T}/\gamma)}{\alpha}\right] \;\geq\; 1 - \frac{\gamma}{2\mathcal{T}}, \tag{4}$$

where the probability is taken (only) over the resamplings in the loop. Now a union bound shows that (3) and the bound from (4) hold for all t, except with probability γ. This finally yields (2). □

The split approach of Algorithm 10 reflects our upcoming rewinding scenario. In particular, conditioning on a "common partial history" $h_t(X_t) = h_t(X_{t-1})$ will correspond to rewinding a simulation up to the t-th "branching point", while $g_t(X_t) = g_t(X_{t-1})$ is a condition we hope the rewound simulation to fulfill. We will be able to sample from $\mathcal{D} | [h_t(\cdot) = h_t(X_{t-1})]$ directly through rewinding, but will then have to condition on $g_t(\cdot) = g_t(X_{t-1})$ by a brute-force **repeat** loop.

4 Adaptive security for the GGM PC-PRF

In this section we use the results on repeated resampling from Section 3 to prove that the PRF construction by Goldreich, Goldwasser and Micali [GGM84b] is adaptively secure as a prefix-constrained pseudorandom function (PC-PRF), based on the security of the underlying PRG.

Definition 6 (GGM PRF). *Given a length-doubling PRG $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ and an input length $d = d(\lambda)$, the GGM PRF $\mathsf{F}_d : \{0,1\}^\lambda \times \{0,1\}^d \to \{0,1\}^\lambda$ with key space $\{0,1\}^\lambda$ is defined as*

$$\mathsf{F}_d(k,x) = k_x \text{ where } k_\varepsilon = k \text{ and } \forall x' \in \{0,1\}^{<d} : k_{x'\|0} \| k_{x'\|1} = \mathsf{G}(k_{x'}).$$

It was noted independently in [Kia+13], [BW13], and [BGI14] that the above PRF construction allows for the use as a prefix-constrained PRF (PC-PRF), with

$$\mathsf{constrain}(k, x') := (x', \mathsf{F}_{|x'|}(k, x')) = (x', k_{x'}) \text{ for } x' \in \{0,1\}^{<d}.$$

For ease of presentation, in the following we will often refer to $k_{x'}$ as the constrained key for x'. The algorithm ceval, on input a constrained key $(x', k_{x'})$ for a prefix x' of x and the string $x = x' \| x'' \in \{0,1\}^d$, then computes k_x as

$$\mathsf{ceval}((x', k_{x'}), x) := \mathsf{constrain}(k_{x'}, x'').$$

4.1 Proving security from PR

We now define the security experiment $\mathrm{Exp}^{\mathsf{GGMPRF}}_{\mathsf{G},\mathcal{A},d}$. Security in the sense of the following definition immediately implies adaptive security of the GGM PC-PRF (see also Remark 1).

Definition 7 (GGMPRF security experiment). *Let* $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ *be a length-doubling PRG, let* $d = d(\lambda)$ *an input length, and let* $\mathcal{A}$ *be a probabilistic adversary. We denote the first half of the output of* G *on input* k *by* $\mathsf{G}_0(k)$*, the second half by* $\mathsf{G}_1(k)$*. The experiment* $\mathrm{Exp}^{\mathsf{GGMPRF}}_{\mathsf{G},\mathcal{A},d}$ *initially samples uniformly at random a challenge bit* $b_{\mathsf{ggmprf}} \leftarrow \{0,1\}$ *and a key* $k_\varepsilon \leftarrow \{0,1\}^\lambda$*. The adversary* $\mathcal{A}$ *can then make the following queries:*

- *$\mathcal{A}$ can adaptively make corruption queries for strings $x \in \{0,1\}^{\leq d}$. This initiates the computation of all so far undefined keys $k_{x'b}$ with $x' <_{\mathsf{pfx}} x$ and $b \in \{0,1\}$ as $k_{x'b} := \mathsf{G}_b(k_{x'})$, and exposes k_x to $\mathcal{A}$.*
- *At any point, $\mathcal{A}$ may stop the game and ask to be challenged on $x^* \in \{0,1\}^d$, and then has to distinguish the real key k_{x^*} (case $b_{\mathsf{ggmprf}} = 0$) from a random key (case $b_{\mathsf{ggmprf}} = 1$). To make the game non-trivial, for the challenge x^* it must hold that no corruption of any prefix of x^* was made throughout the game.*

The output of the experiment is 1 if $\mathcal{A}$ *correctly guesses the bit* b_{ggmprf}*, and 0 otherwise. We define the advantage of* $\mathcal{A}$ *in this game as*

$$\mathrm{Adv}^{\mathsf{GGMPRF}}_{\mathsf{G},\mathcal{A},d}(\lambda) := \Pr[\mathrm{Exp}^{\mathsf{GGMPRF}}_{\mathsf{G},\mathcal{A},d}(\lambda) = 1] - 1/2.$$

We say that GGMPRF *security holds (for* G *and* d*) if* $\mathrm{Adv}^{\mathsf{GGMPRF}}_{\mathsf{G},\mathcal{A},d}(\lambda)$ *is negligible for every probabilistic polynomial-time* $\mathcal{A}$*.*

One can view this security experiment as a game on a binary tree of depth d as defined in Section 2.1, where the adversary can adaptively compromise labels k_x. For security, we require that keys that cannot be computed trivially from compromised keys should remain pseudorandom.

Remark 1. We note that we consider adversaries that make their challenge query as the last query. This is not a restriction as any adaptive adversary can be transformed into such an adversary with only d additional constrained key queries, using the following reduction: All queries and responses until the challenge query are forwarded. Once the adversary submits the challenge query, the reduction queries all constrained keys on the co-path before forwarding the challenge query and its response. To answer any future constrained key queries, the reduction uses the previously queried constrained keys on the co-path.

To see that security of the GGM PRF as a PC-PRF follows from our results on the GGMPRF security experiment, note that a reduction can answer adversarial constrained key queries and PRF evaluation queries in the $\mathsf{PC\text{-}PRF}$ security experiment for the GGM PRF by making corresponding corruption queries in the GGMPRF security experiment. In particular, this means that for any adversary $\mathcal{A}$ that has advantage $\mathrm{Adv}^{\mathsf{GGMPRF}}_{\mathsf{G},\mathcal{A},d}(\lambda)$, runs in time $t_{\mathcal{A}}$, and makes $\mathcal{Q}_{\mathsf{corrupt}}$ constrained key queries, there exists an adversary $\mathcal{B}$ that runs in time $t_{\mathcal{B}}$ roughly equal[10] to $t_{\mathcal{A}}$ with

$$\mathrm{Adv}^{\mathsf{PC\text{-}PRF}}_{\mathsf{F}_d,\mathcal{A}}(\lambda) = \mathrm{Adv}^{\mathsf{GGMPRF}}_{\mathsf{G},\mathcal{A},d}(\lambda)$$

and makes $\mathcal{Q}'_{\mathsf{corrupt}} \leq \mathcal{Q}_{\mathsf{corrupt}} + d$ constrained key queries.

Our strategy. Let us fix a length-doubling PRG G and a depth/input length $d = d(\lambda)$. Let us first consider a *selective* setting where an adversary $\mathcal{A}$ has to commit to the challenge x^* in the beginning of the game. For this setting, we can bound the success probability of any PPT adversary $\mathcal{A}$ by a sequence of $d+1$ hybrid games where in the ith hybrid game, the first i PRG evaluations on the path from the root to x^* are replaced by random sampling, i.e. the keys $k_{x\|0}, k_{x\|1}$ for all $x \leq_{\mathsf{pfx}} x^*$ with $|x| < i$ are sampled uniformly at random instead of computing $\mathsf{G}(k_x)$. For each $i \in [d]$, one can then prove that games $i-1$ and i are indistinguishable based on the security of the PRG G. Furthermore, since in game d, the key k_{x^*} is sampled independently and uniformly at random, the cases $b_{\mathsf{ggmprf}} = 0$ and $b_{\mathsf{ggmprf}} = 1$ are information-theoretically indistinguishable, hence the advantage of $\mathcal{A}$ is 0 in this game. We thus obtain an upper bound on $\mathcal{A}$'s advantage in the selective GGMPRF experiment in terms of PR security of the PRG G, with a security loss linear in d.

Also in the adaptive setting, where $\mathcal{A}$ can make its choices on the fly, we will bound $\mathcal{A}$'s advantage to win the GGMPRF game through a similar hybrid argument. Again, we will start with the original GGMPRF game above and apply a number of successive changes until finally $\mathcal{A}$'s view is independent of the challenge bit b_{ggmprf}. Since we make a liberal use of rewindings, it will be helpful to formalize $\mathcal{A}$'s view:

Definition 8 (Adversarial view). *In a run of the experiment* $\mathrm{Exp}^{\mathsf{GGMPRF}}_{\mathsf{G},\mathcal{A},d}$ *from Definition 7, we define $\mathcal{A}$'s view* $\mathsf{view}_{\mathcal{A}}$ *in this run as a sequence* $(\mathsf{ev}_1, \ldots, \mathsf{ev}_\ell)$ *of events, where each* ev_i *can be one of the following:*

Query. *One of $\mathcal{A}$'s queries (without reply), either of the form* $(\mathsf{corrupt}, x)$ *for a corruption query, or* $(\mathsf{challenge}, x^*)$.

[10]By "roughly equal", we mean that $\mathcal{B}$ runs $\mathcal{A}$ only once, but as discussed with up to d added oracle queries and some additional $\mathsf{constrain}$ operations.

New keys. *Every time new keys* $k_{x\|0}, k_{x\|1}$ *are defined, right before that, a corresponding* (PRG, x) *event is appended to* view*. Concretely, a query* $(\mathsf{corrupt}, x)$ *or* $(\mathsf{challenge}, x^*)$ *in* view *automatically causes also entries* (PRG, x') *for all proper prefixes* x' *of* x *for which no* PRG *query has been issued yet, to be appended immediately after that* $(\mathsf{corrupt}, x)$ *entry. Entries* (PRG, x) *defined at the same time are ordered in* view *with keys closer to the root (i.e., with shorter* x*) first.*

Corrupted key. *A key* (key, x, k_x) *as a response to a corruption query.*

Challenge key. *The response to the final challenge query, in the form* $(\mathsf{challenge}, x^*, k)$ *(i.e., depending on* b_{ggmprf} *with either* k *being the real key* k_{x^*} *or a random value). This event comes after the corresponding* (PRG, x) *events which are triggered by the challenge query.*

Decision bit. *The final output bit* $b_{\mathcal{A}}$ *of* $\mathcal{A}$*, in the form* $(\mathsf{guess}, b_{\mathcal{A}})$*. This event is the last in* view*, and we may write* $\mathbf{out}_{\mathcal{A}}(\mathsf{view})$ *to denote that bit* $b_{\mathcal{A}}$*.*

We are now ready to formulate and prove our main result:

Theorem 1. *Let* $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ *be a PRG. Then*
- *for every* GGMPRF *adversary* $\mathcal{A}$ *that runs in time* $t_{\mathcal{A}}$ *and makes at most* $\mathcal{Q}_{\mathsf{corrupt}}$ *corrupt queries,*
- *for every* GGMPRF *depth* d *and every* $\gamma \in (0,1]$*,*

there is a PR *adversary* $\mathcal{B}$ *that runs in time* $t_{\mathcal{B}}$*, makes at most* $\mathcal{Q}_{\mathcal{B}}$ *oracle queries, and for which*

$$\mathrm{Adv}^{\mathsf{PR}}_{\mathsf{G},\mathcal{B}}(\lambda) \;\geq\; \frac{1}{2d} \cdot \left(\mathrm{Adv}^{\mathsf{GGMPRF}}_{\mathsf{G},\mathcal{A},d}(\lambda) - \gamma\right), \tag{5}$$

where

$$t_{\mathcal{B}} \;\lessapprox\; \left(2 \cdot \ln\left(2 \cdot \mathcal{T}/\gamma\right) \cdot \mathcal{T}^4/\gamma\right) \cdot t_{\mathcal{A}} \quad \text{and} \quad \mathcal{Q}_{\mathcal{B}} \;\leq\; 2 \cdot \ln\left(2 \cdot \mathcal{T}/\gamma\right) \cdot \mathcal{T}^3/\gamma \tag{6}$$

with $\mathcal{T} \leq ((d+2) \cdot \mathcal{Q}_{\mathsf{corrupt}} + 2)$*.*

Before we proceed to a proof, we notice that Theorem 1 implies asymptotic security when setting γ accordingly:

Corollary 1 (G secure $\Rightarrow$ GGM PRF secure PC-PRF). *If* G *is a secure PRG (as in Definition 1) and* $d = d(\lambda)$ *is a polynomial, then the GGM PRF* F_d *is an indistinguishable PC-PRF (as in Definition 3).*

Proof of Corollary 1. Assume for contradiction that there is a polynomial-time adversary $\mathcal{A}'$ against the PC-PRF indistinguishability with non-negligible advantage. By Remark 1, this immediately yields a polynomial-time GGMPRF adversary $\mathcal{A}$ with (the same) non-negligible advantage $\varepsilon_{\mathcal{A}} := \mathrm{Adv}^{\mathsf{GGMPRF}}_{\mathsf{G},\mathcal{A},d}(\lambda)$. Since $\varepsilon_{\mathcal{A}}$ is non-negligible, there exists a polynomial p such that for infinitely many values of λ, we have $\varepsilon_{\mathcal{A}} \geq 1/p(\lambda)$.

Now set $\gamma = 1/(2p(\lambda))$ and invoke Theorem 1. We obtain a PR adversary $\mathcal{B}$ with (by (6)) polynomial runtime and non-negligible advantage

$$\mathrm{Adv}^{\mathsf{PR}}_{\mathsf{G},\mathcal{B}}(\lambda) \;\overset{(5)}{\geq}\; \frac{1}{2d} \cdot \left(\varepsilon_{\mathcal{A}} - \gamma\right) \;\overset{(*)}{\geq}\; \frac{1}{2d} \cdot \left(\frac{1}{p(\lambda)} - \gamma\right) \;=\; \frac{1}{4d \cdot p(\lambda)},$$

where $(*)$ holds (only) for infinitely many λ. □

Proof of Theorem 1. Fix $\mathcal{A}$ and d. In the following, we will consider a number of hybrid games, with Game ggmprf being the original GGMPRF experiment. Denoting with $\mathbf{out}_i$ the output of Game i, we trivially get

$$\Pr[\mathbf{out}_{\mathsf{ggmprf}} = 1] \;=\; \mathrm{Adv}^{\mathsf{GGMPRF}}_{\mathsf{G},\mathcal{A},d}(\lambda) + 1/2. \tag{7}$$

Moving on, we will formulate Game i (for $0 \leq i \leq d$) in a (for us) convenient way, see Algorithm 11. This formulation outsources the bulk of the game into the sampling of $\mathcal{A}$'s view view from a suitable distribution $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{ggmprf}}}$. In our upcoming

Algorithm 11: Game i, with the bulk of the work outsourced into the sampling from $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{ggmprf}}}$.

```
1 $b_{\mathsf{ggmprf}} \leftarrow \{0,1\}$
2 $\mathsf{view} \leftarrow \mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{ggmprf}}}$                // view has the format from Definition 8
3 return $[b_{\mathsf{ggmprf}} = \mathbf{out}_{\mathcal{A}}(\mathsf{view})]$          // returns 1 iff $b_{\mathsf{ggmprf}} = \mathbf{out}_{\mathcal{A}}(\mathsf{view})$
```

refinements, we will only change $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{ggmprf}}}$ and investigate the effects on $\mathbf{out}_i$.

The distributions $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{ggmprf}}}$. To define the distribution $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{ggmprf}}}$ for Game i with $i \in [d]_0$, we use the following notation:

- $\mathcal{D}^{\mathsf{view}}_{\mathsf{ggmprf},b_{\mathsf{ggmprf}}}$ is the distribution of $\mathcal{A}$-views (as in Definition 8) that is induced by running the GGMPRF experiment with challenge bit b_{ggmprf} (that decides whether $\mathcal{A}$ is challenged with k_{x^*} or a random key).
- $\mathsf{len}(\mathsf{view})$ is the length of a given $\mathcal{A}$-view view (measured in events).
- $\mathsf{pfx}_t(\mathsf{view})$ outputs the prefix of view up to (and including) the t-th event (as defined in Section 2.1).
- $\mathsf{lastpre}_t(\mathsf{view})$ on input $\mathsf{view} = (\mathsf{ev}_1, \ldots, \mathsf{ev}_{\mathcal{T}})$ outputs the largest index $t' \leq t$ such that event $\mathsf{ev}_{t'}$ defines a key on the path from the root to x^*, i.e.,

$$\mathsf{lastpre}_t(\mathsf{view}) := \max\left(\left\{ t' \;\middle|\; \begin{array}{c} \mathsf{ev}_{t'} = (\mathsf{PRG}, x) \\ \wedge\, t' \leq t \;\wedge\; x <_{\mathsf{pfx}} x^* \end{array} \right\} \cup \{0\}\right).$$

- $B \in \mathbb{N}$ is a bound on the number of repetitions of Lines 6 to 13 in Algorithm 12 for each t. In case of B unsuccessful repetitions for one t, the whole algorithm outputs $\perp$. We will fix a suitable value for B later.

Now consider Algorithm 12. Our distribution $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{ggmprf}}}$ will be defined almost like $\mathcal{D}^{\mathsf{view}}_{\mathsf{ggmprf},b_{\mathsf{ggmprf}}}$ (i.e., like $\mathcal{A}$'s view in a GGMPRF run), but will additionally replace PRG evaluations by random sampling as indicated by index i and use rewindings at every step. Concretely, fix an i and consider Algorithm 12, which programmatically defines $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{ggmprf}}}$ as its output.

For an adversary $\mathcal{A}$ with view $\mathsf{view} = (\mathsf{ev}_1, \ldots, \mathsf{ev}_{\mathcal{T}})$ we denote by *rewinding* the adversary to time t the cutting-off of the view at point $t-1$, i.e. $(\mathsf{ev}_1, \ldots, \mathsf{ev}_{t-1})$ and resetting the adversary to the state it had directly before

ev_t. (By keeping track of $\mathcal{A}$'s state throughout our rewindings, this will always be possible.)

We say we *resample* from point t (after rewinding $\mathcal{A}$ to t) if we rerun $\mathcal{A}$ from t onwards, using fresh challenger random coins from that point onwards. In some cases, we will also rerun $\mathcal{A}$ with a specific replacement (e.g., an embedded computational challenge) in ev_t (if ev_t contains an answer to one of $\mathcal{A}$'s previous queries). The view resulting from rewinding to t and then resampling from point t is $\mathsf{view}' = (\mathsf{ev}_1, \dots, \mathsf{ev}_{t-1}, \mathsf{ev}'_t, \dots, \mathsf{ev}'_{\mathcal{T}'})$.

Algorithm 12: Sampler for $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{ggmprf}}}$

```
   Input: $i \in \{0,\dots,d\}, b_{\mathsf{ggmprf}} \in \{0,1\}, B \in \mathbb{N}$ // $\mathsf{len}, \mathsf{lastpre}_{i,t}, B$ described in
      proof
1  $\mathsf{view}_0 \leftarrow \mathcal{D}^{\mathsf{view}}_{\mathsf{ggmprf},b_{\mathsf{ggmprf}}}$
2  $\mathcal{T} := \mathsf{len}(\mathsf{view}_{\mathsf{GGMPRF}})$          // Length of $\mathsf{view}_{\mathsf{GGMPRF}}$ (in entries)
3  for $t := 1$ to $\mathcal{T}$ do
4      Write $\mathsf{view}_{t-1} = (\mathsf{ev}_{t-1,1}, \dots, \mathsf{ev}_{t-1,\mathcal{T}})$
5      repeat            // Output $\perp$ if $B$ repetitions fail for this $t$
6          Rewind adversary to point $t$
7          if $\mathsf{ev}_{t-1,t} = (\mathsf{PRG}, x)$ with $|x| \le i$ and $\mathsf{lastpre}_t(\mathsf{view}_{t-1}) = t$ then
            // Checks if $\mathsf{ev}_{t-1,t}$ defines a PRG evaluation to be replaced
            by random
8              Sample fresh $k_{x\|0}, k_{x\|1} \leftarrow \{0,1\}^\lambda$     // Fresh independent keys
9              Resample from point $t+1$ to obtain
                $\mathsf{view}_t = (\mathsf{ev}_{t-1,0}, \dots, \mathsf{ev}_{t-1,t-1}, \mathsf{ev}_{t-1,t}, \mathsf{ev}_{t,t+1}, \dots, \mathsf{ev}_{t,\tau})$
10         else
11             Resample from point $t$ to obtain
                $\mathsf{view}_t = (\mathsf{ev}_{t-1,0}, \dots, \mathsf{ev}_{t-1,t-1}, \mathsf{ev}_{t,t}, \dots, \mathsf{ev}_{t,\tau})$
12         end
13     until $\mathsf{lastpre}_t(\mathsf{view}_t) = \mathsf{lastpre}_t(\mathsf{view}_{t-1})$ and $\mathsf{len}(\mathsf{view}_t) = \mathsf{len}(\mathsf{view}_{t-1})$
14 end
15 return $\mathsf{view}_{\mathcal{T}}$
```

Having defined our hybrid distributions $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{ggmprf}}}$, we will additionally consider the distribution $\widetilde{\mathcal{D}}^{\mathsf{view}}_{0,b_{\mathsf{ggmprf}}}$ which is defined as the output of a variant of Algorithm 12 for $i = 0$ without a bound B on the runtime. (Hence, $\widetilde{\mathcal{D}}^{\mathsf{view}}_{0,b_{\mathsf{ggmprf}}}$ will not be efficiently sampleable in general.) We will first show that the distribution $\widetilde{\mathcal{D}}^{\mathsf{view}}_{0,b_{\mathsf{ggmprf}}}$ coincides with the distribution of views in Game ggmprf. Here we will use our results from Section 3, namely Lemma 5, for the distribution $\mathcal{D} = \mathcal{D}^{\mathsf{view}}_{\mathsf{ggmprf},b_{\mathsf{ggmprf}}}$ where functions h_t on input view will output the $(t-1)$-sized prefix of view, and resampling conditioned on $h_t(\mathsf{view})$ simply means rewinding and rerunning from point t. The stopping conditions $g_t(\mathsf{view})$ will preserve (1) the value of $\mathsf{lastpre}_t(\mathsf{view})$, and (2) the length of view. Intuitively, preserving

$$\begin{tikzcd}
\text{Game } \mathsf{ggmprf} \arrow[r, dash, "\equiv"] & \widetilde{\mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{ggmprf}}}} \arrow[d, dash, "\overset{\mathrm{s}}{\approx}_{\gamma}"] & & & & \\
 & \mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{ggmprf}}} \arrow[r, dash, "\overset{\mathrm{c}}{\approx}"] & \mathcal{D}^{\mathsf{view}}_{1,b_{\mathsf{ggmprf}}} \arrow[r, dash, "\overset{\mathrm{c}}{\approx}"] & \mathcal{D}^{\mathsf{view}}_{2,b_{\mathsf{ggmprf}}} \arrow[r, dash, "\overset{\mathrm{c}}{\approx}"] & \cdots \arrow[r, dash, "\overset{\mathrm{c}}{\approx}"] & \mathcal{D}^{\mathsf{view}}_{d,b_{\mathsf{ggmprf}}}
\end{tikzcd}$$

Fig. 6: Sequence of hybrids. Perfect indistinguishability ($\equiv$) is shown in Proposition 1, statistical distance ($\overset{\mathrm{s}}{\approx}_{\gamma}$) is shown in Proposition 2, and computational indistinguishability ($\overset{\mathrm{c}}{\approx}$) is shown in Proposition 3.

$\mathsf{lastpre}_t(\mathsf{view})$ implies that "PRG embedding slots" along the path to the challenge x^* defined prior to point t remain the same. This implies that no preimages of previously embedded PRG images have to be revealed, and the rewinding did not "undo" any of the progress made so far.

Again using our results from Section 3, namely Lemma 6 with similar interpretation as above, we will then choose the bound B such that the probability of an abort in $\mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{ggmprf}}}$ can be bounded by γ. Then we will show that $\mathcal{A}$ has no advantage in Game d since the view sampled according to $\mathcal{D}^{\mathsf{view}}_{d,b_{\mathsf{ggmprf}}}$ is independent of b_{ggmprf}. Finally, we will argue that $\mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{ggmprf}}}$ is computationally indistinguishable from $\mathcal{D}^{\mathsf{view}}_{d,b_{\mathsf{ggmprf}}}$ by the pseudorandomness of G. Combining these results will allow us to conclude the proof. Our path along this sequence of hybrids can be seen in Fig. 6.

Proposition 1. $\mathcal{D}^{\mathsf{view}}_{\mathsf{ggmprf},b_{\mathsf{ggmprf}}} \equiv \widetilde{\mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{ggmprf}}}}$.

Proof. This follows from Lemma 5, where $\mathcal{D} = \mathcal{D}^{\mathsf{view}}_{\mathsf{ggmprf},b_{\mathsf{ggmprf}}}$, $h_t(\mathsf{view}_s) = (\mathsf{ev}_{s,1}, \ldots, \mathsf{ev}_{s,t-1})$, and $g_t(\mathsf{view}_s) = (\mathsf{lastpre}_t(\mathsf{view}_s), \mathsf{len}(\mathsf{view}_s))$. We note that for $i = 0$, the **if** on Line 7 never returns true, and thus the sampling procedure always enters the **else** branch which behaves just as in Lemma 5. □

Proposition 2 (Abort probability). *For*

$$B := 2 \cdot \ln\left(2 \cdot ((d+2) \cdot \mathcal{Q}_{\mathsf{corrupt}} + 2)/\gamma\right) \cdot \left((d+2) \cdot \mathcal{Q}_{\mathsf{corrupt}} + 2\right)^3/\gamma,$$

we have $\Pr[\bot \leftarrow \mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{lkh}}}] \leq \gamma$.

Proof. To prove this claim, we consider the process of sampling from $\mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{lkh}}}$ according to Algorithm 12 and bound the probability that any of the iterations in the "**for**" loop runs the "**repeat**" loop more than B times.

By Lemma 6, with $g_t(\mathsf{view}) = (\mathsf{lastpre}_t(\mathsf{view}), \mathsf{len}(\mathsf{view}))$ and $h_t(\mathsf{view}) = (\mathsf{ev}_1, \ldots, \mathsf{ev}_t)$ for $\mathsf{view} = (\mathsf{ev}_1, \ldots, \mathsf{ev}_{\mathsf{len}(\mathsf{view})})$, it holds that for any $\gamma \in (0,1]$ (thus in particular the γ from the theorem statement)

$$\Pr[\, \forall t \in [\mathcal{T}] : T^{\mathsf{rep}}_t \leq 2\mathcal{T} \cdot \ln(2\mathcal{T}/\gamma) \cdot |\mathcal{Y}|/\gamma \,] \;\geq\; 1 - \gamma \tag{8}$$

where T_t^{rep} denotes the number of runs of the "**repeat**" loop in the t-th iteration of the "**for**" loop. We note that

$$\mathsf{len}(\mathsf{view}) \leq \underbrace{\mathcal{Q}_{\mathsf{corrupt}} + 1}_{\text{Query Events}} + \underbrace{d \cdot \mathcal{Q}_{\mathsf{corrupt}}}_{\text{PRG Events}} + \underbrace{\mathcal{Q}_{\mathsf{corrupt}} + 1}_{\text{Corr./Chal. Key Events}}$$

for any view resulting from a run of an adversary that makes at most $\mathcal{Q}_{\mathsf{corrupt}}$ constrained key queries. This means that $\mathsf{len}(\mathsf{view})$ can take values up to $\mathcal{T}_{\max} = (d+2) \cdot \mathcal{Q}_{\mathsf{corrupt}} + 2$. Furthermore, $\mathsf{lastpre}_t(\mathsf{view})$ takes values from 0 to $\mathsf{len}(\mathsf{view})$. Thus, we can bound the size of the range $\mathcal{Y}$ of the g_t with $|\mathcal{Y}| \leq ((d+2) \cdot \mathcal{Q}_{\mathsf{corrupt}} + 2)^2$.

Plugging this into (8) yields

$$\Pr\left[\ \forall t \in [\mathcal{T}] : T_t^{\mathsf{rep}} \leq 2\mathcal{T} \cdot \ln(2\mathcal{T}/\gamma) \cdot ((d+2) \cdot \mathcal{Q}_{\mathsf{corrupt}} + 2)^2 / \gamma\ \right] \ \geq\ 1 - \gamma. \quad (9)$$

Thus, using the bound for $\mathsf{len}(\mathsf{view})$ for $\mathcal{T}$ again, i.e. $\mathcal{T} \leq (d+2) \cdot \mathcal{Q}_{\mathsf{corrupt}} + 2$, gives $\Pr[\ \forall t \in [\mathcal{T}] : T_t^{\mathsf{rep}} \leq B\ ] \ \geq\ 1 - \gamma$. which yields the claim. □

Proposition 3 ($\mathsf{PR} \Rightarrow \mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{ggmprf}}} \overset{c}{\approx} \mathcal{D}^{\mathsf{view}}_{d,b_{\mathsf{ggmprf}}}$). *Let B be as in Claim 2. If* G *is* PR *secure, then the distributions $\mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{ggmprf}}}$ and $\mathcal{D}^{\mathsf{view}}_{d,b_{\mathsf{ggmprf}}}$ are computationally indistinguishable. More precisely, there exists a* PR *adversary $\mathcal{C}$ that runs in time $t_{\mathcal{C}}$ and makes $\mathcal{Q}_{\mathcal{C}}$ oracle queries, such that*

$$\mathrm{Adv}^{\mathsf{PR}}_{\mathsf{G},\mathcal{C}}(\lambda) \ =\ \frac{1}{2d} \cdot \left(\mathrm{Adv}^{\mathsf{GGMPRF},0}_{\mathsf{G},\mathcal{A},d}(\lambda) - \mathrm{Adv}^{\mathsf{GGMPRF},d}_{\mathsf{G},\mathcal{A},d}(\lambda)\right) \quad (10)$$

$$t_{\mathcal{C}} \ \lessapprox\ \left(2 \cdot \ln\left(2 \cdot \mathcal{T}_{\max}/\gamma\right) \cdot \mathcal{T}_{\max}^4/\gamma\right) \cdot t_{\mathcal{A}} \quad \text{and} \quad \mathcal{Q}_{\mathcal{C}} \ \leq\ 2 \cdot \ln\left(2 \cdot \mathcal{T}/\gamma\right) \cdot \mathcal{T}^3/\gamma \quad (11)$$

with $\mathcal{T} \leq \mathcal{T}_{\max} = (d+2) \cdot \mathcal{Q}_{\mathsf{corrupt}} + 2$.

Proof. To generate a sample $\mathsf{view}_{\mathcal{T}}$, our PR adversary $\mathcal{C}$ modifies the procedure of Algorithm 12 by first sampling an index $i^* \leftarrow [d]$ and a bit $b_{\mathsf{pr}} \leftarrow \{0,1\}$ uniformly at random, and then embedding a PR challenge in the "**if**" clause in the "**repeat**" loop, see Algorithm 13. $\mathcal{C}$ outputs 0 if $\mathcal{A}$ succeeds and 1 else.

Note that the key for node $\mathsf{pfx}_{i^*-1}(x^*)$ is sampled freshly and uniformly at random. Thus, we have that for $b_{\mathsf{pr}} = 0$ and $i^* = i$ the modified algorithm samples from exactly the same distribution as Algorithm 12 on input $i-1$ (and same $b_{\mathsf{ggmprf}} \in \{0,1\}, B \in \mathbb{N}$), and for $b_{\mathsf{pr}} = 1$ and $i^* = i$ from the same distribution as Algorithm 12 on input i (and same $b_{\mathsf{ggmprf}} \in \{0,1\}, B \in \mathbb{N}$). We obtain for the advantage of $\mathcal{C}$:

$$\begin{aligned}
\mathrm{Adv}^{\mathsf{PR}}_{\mathsf{G},\mathcal{C}}(\lambda) &= \Pr[b_{\mathcal{C}} = b_{\mathsf{pr}}] - \frac{1}{2} \\
&= \frac{1}{2} \cdot \left(\Pr[b_{\mathcal{C}} = 0 \mid b_{\mathsf{pr}} = 0] - \Pr[b_{\mathcal{C}} = 0 \mid b_{\mathsf{pr}} = 1]\right) \\
&= \frac{1}{2} \cdot \frac{1}{d} \cdot \sum_{i \in [d]} \left(\Pr\left[b_{\mathcal{C}} = 0 \,\middle|\, \begin{array}{c} b_{\mathsf{pr}} = 0 \\ \wedge\ i^* = i \end{array}\right] - \Pr\left[b_{\mathcal{C}} = 0 \,\middle|\, \begin{array}{c} b_{\mathsf{pr}} = 1 \\ \wedge\ i^* = i \end{array}\right]\right)
\end{aligned}$$

$$= \frac{1}{2d} \cdot \sum_{i \in [d]} (\Pr[\mathbf{out}_{i-1} = 1] - \Pr[\mathbf{out}_i = 1])$$
$$= \frac{1}{2d} \cdot \left(\mathrm{Adv}^{\mathsf{GGMPRF},\,0}_{\mathsf{G},\mathcal{A},d}(\lambda) - \mathrm{Adv}^{\mathsf{GGMPRF},\,d}_{\mathsf{G},\mathcal{A},d}(\lambda)\right).$$

Algorithm 13: Variant of Algorithm 12 for sampling from $\mathcal{D}^{\mathsf{view}}_{i^*-1+b_{\mathsf{pr}},b_{\mathsf{ggmprf}}}$ given oracle access to a PR challenger with challenge bit b_{pr}. The functions $\mathsf{len}, \mathsf{lastpre}_t$ are as described in the proof, $B \in \mathbb{N}$ is as in Proposition 2.

```
1  $i^* \leftarrow \{1,\dots,d\}, b_{\mathsf{ggmprf}} \leftarrow \{0,1\}$
2  $\mathsf{view}_0 \leftarrow \mathcal{D}^{\mathsf{view}}_{\mathsf{ggmprf},b_{\mathsf{ggmprf}}}$
3  $\mathcal{T} := \mathsf{len}(\mathsf{view}_0)$                    // Length of $\mathsf{view}_{\mathsf{GGMPRF}}$ (in entries)
4  for $t := 1$ to $\mathcal{T}$ do
5      Write $\mathsf{view}_{t-1} = (\mathsf{ev}_{t-1,1},\dots,\mathsf{ev}_{t-1,\mathcal{T}})$
6      repeat              // Output $\perp$ if $B$ repetitions fail for this $t$
7          Rewind adversary to point $t$
8          if $\mathsf{ev}_{t-1,t} = (\mathsf{PRG}, x)$ with $|x| \le i^*$ and $\mathsf{lastpre}_t(\mathsf{view}_{t-1}) = t$ then
               // Checks if $\mathsf{ev}_{t,t}$ defines a PRG evaluation to be replaced by random
9              if $|x| = i^*$ then
10                 Request fresh PR challenge $(k_0^*, k_1^*)$ from PR challenger
                    // Fresh PR challenge
11                 Set $(k_{x\|0}, k_{x\|1}) := (k_0^*, k_1^*)$
12             else
13                 Sample fresh $k_{x\|0}, k_{x\|1} \leftarrow \{0,1\}^\lambda$      // Fresh independent keys
14             end
15             Resample from point $t+1$ to obtain
                 $\mathsf{view}_t = (\mathsf{ev}_{t-1,0},\dots,\mathsf{ev}_{t-1,t-1},\mathsf{ev}_{t-1,t},\mathsf{ev}_{t,t+1}\dots,\mathsf{ev}_{t,\tau})$
16         else
17             Resample from point $t$ to obtain
                 $\mathsf{view}_t = (\mathsf{ev}_{t-1,0},\dots,\mathsf{ev}_{t-1,t-1},\mathsf{ev}_{t,t},\dots,\mathsf{ev}_{t,\tau})$
18         end
19     until $\mathsf{lastpre}_t(\mathsf{view}_t) = \mathsf{lastpre}_t(\mathsf{view}_{t-1})$ and $\mathsf{len}(\mathsf{view}_t) = \mathsf{len}(\mathsf{view}_{t-1})$
20 end
21 return $\mathsf{view}_{\mathcal{T}}$
```

$\mathcal{C}$ runs $\mathcal{A}$ at most $\mathcal{T} \cdot B$ times. Bounding $\mathcal{T} = \mathsf{len}(\mathsf{view}_0)$ by $\mathcal{T}_{\max} = (d+2) \cdot \mathcal{Q}_{\mathsf{corrupt}} + 2$ (see proof of Claim 2) and plugging in $B = \frac{2\mathcal{T} \cdot \ln(2\mathcal{T}/\gamma) \cdot ((d+2) \cdot \mathcal{Q}_{\mathsf{corrupt}}+2)^2}{\gamma}$ leads to the claimed bound on $t_{\mathcal{C}}$. (We assume the time complexity of random sampling, PR oracle calls and PRG evaluations to be significantly smaller than $t_{\mathcal{A}}$ and thus neglect the corresponding terms in our bound.)

For the upper bound on the number of oracle calls $\mathcal{Q}_{\mathcal{C}}$, note that for each possible choice of i^* there is only one t such that $\mathsf{ev}_{t-1,t} = (\mathsf{PRG}, x)$ with $|x| = i^*$ and $\mathsf{lastpre}_t(\mathsf{view}_{t-1}) = t$. Thus, the inner **if** clause will apply only in one of the **for** iterations, which implies that there are as many PR calls as there are iterations of the **repeat** loop for that t. Hence, the reduction makes at most B calls to the PR oracle. □

Proposition 4. *$\mathcal{D}^{\mathsf{view}}_{d,b_{\mathsf{ggmprf}}}$ and b_{ggmprf} are independent, so* $\mathrm{Adv}^{\mathsf{GGMPRF},\,d}_{\mathsf{G},\mathcal{A},d}(\lambda) = 0$.

Proof. Recall that b_{ggmprf} is only used when responding to the challenge query, which by assumption is the last query the adversary makes. Hence, neither the abort probability nor any of the events in $\mathsf{view}_{\mathcal{T}}$ before the very last events $(\mathsf{challenge}, x^*, k)$ and $(\mathsf{guess}, b_{\mathcal{A}})$ depend on b_{ggmprf}. The latter also implies that the values for $\mathsf{lastpre}$ and len are independent of b_{ggmprf} for all t. As the last two events of the view (that are the only ones carrying information about b_{ggmprf}) are cut off when rewinding and resampling, no information about b_{ggmprf} is carried from view_{t-1} to view_t for any t. Furthermore, in the final view $\mathsf{view}_{\mathcal{T}}$ the challenge key k_{x^*} is sampled independently and uniformly at random, hence k has the same distribution for both cases $b_{\mathsf{ggmprf}} = 0$ and $b_{\mathsf{ggmprf}} = 0$. Thus, $\mathcal{A}$ has no advantage in distinguishing k_{x^*} from a random independent key. □

To finish the proof of the theorem, it only remains to combine the above claims. In particular, we define the adversary $\mathcal{B}$ exactly as $\mathcal{C}$ from Proposition 3. The bound on the runtime of $\mathcal{B}$ follows immediately and for the advantage of $\mathcal{B}$ we have

$$
\begin{aligned}
\mathrm{Adv}^{\mathsf{PR}}_{\mathsf{G},\mathcal{B}}(\lambda) =& \frac{1}{2d} \cdot \left(\mathrm{Adv}^{\mathsf{GGMPRF},\,0}_{\mathsf{G},\mathcal{A},d}(\lambda) - \mathrm{Adv}^{\mathsf{GGMPRF},\,d}_{\mathsf{G},\mathcal{A},d}(\lambda)\right) \\
\geq \quad & \frac{1}{2d} \cdot \left(\mathrm{Adv}^{\widetilde{\mathsf{GGMPRF},\,0}}_{\mathsf{G},\mathcal{A},d}(\lambda) - \mathrm{Adv}^{\mathsf{GGMPRF},\,d}_{\mathsf{G},\mathcal{A},d}(\lambda) - \gamma\right) \\
\geq \quad & \frac{1}{2d} \cdot \left(\mathrm{Adv}^{\mathsf{GGMPRF}}_{\mathsf{G},\mathcal{A},d}(\lambda) - \mathrm{Adv}^{\mathsf{GGMPRF},\,d}_{\mathsf{G},\mathcal{A},d}(\lambda) - \gamma\right) \\
\geq \quad & \frac{1}{2d} \cdot \left(\mathrm{Adv}^{\mathsf{GGMPRF}}_{\mathsf{G},\mathcal{A},d}(\lambda) - \gamma\right).
\end{aligned}
$$

□

5 Adaptive security for LKH

Overview. The main application we have in mind in this section is a multicast key distribution protocol called the Logical Key Hierarchy (LKH) [WHA98; WGL00; Can+99]; more precisely, we consider the rectified version by Panjwani [Pan07]. Our strategy can easily be generalized to minor modifications of LKH and therefore we do not focus on specific implementation details. Rather, we provide a very brief high-level description of the protocol as well as the security guarantees we aim to guarantee. More broadly, we believe that our results provide the core techniques to prove adaptive security also for multicast key agreement as defined in [BDT22], as well as (various versions of) the related "TreeKEM" protocol [BBR18; Kle+21] for (public-key) continuous group key agreement (CGKA).

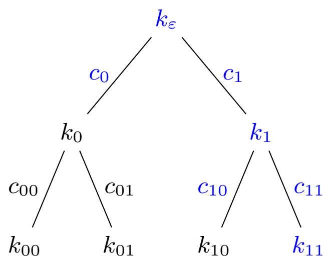
Multicast key distribution (MKD). A protocol for multicast key distribution (MKD, see [Pan07]) is a server-aided secret-key protocol that enables a dynamically changing group of users to securely communicate over a broadcast channel. In an initial registration step, it is assumed that each user establishes a secret key with the server; this key infrastructure setup is however outside the protocol specification. The server then uses these shared secret keys to communicate a group key to the current set of user. Upon a join/leave request, the server refreshes the group key and sends rekey messages to the new set of users, which allow each user to derive the new group key. In the security experiment, the adversary can request join and leave operations for arbitrary users fully adaptively and learns all keys of removed users. Finally, it can request a challenge and in return obtains either the real group key or a random independent key.

Logical key hierarchy (LKH). A trivial MKD protocol would be to simply encrypt a freshly sampled group key to all current users after each membership change. However, for large groups this does not scale well, as it requires a linear number of encryptions. A smarter approach is taken in the Logical Key Hierarchy (LKH) protocol, as proposed in [WHA98; WGL00; Can+99]. We will consider the rectified version of LKH by Panjwani [Pan07]: LKH is based on a binary tree structure, where each node is associated with a secret key k_x and edges represent secret-key encryptions $c_{x\|b}$ of the parent key k_x under the child key $k_{x\|b}$ (see binary tree notation in Section 2.1). The keys associated to leaves in the tree belong to members participating in the multicast key distribution, the key k_ε associated to the root is used as the group key. Users can be added to or removed from the group, which leads to a state update where all the keys and ciphertexts associated with nodes and edges on the path from the user's leaf to the root are refreshed (except for the edge attached to the leaf in case of a remove), and also the edges connecting these nodes to co-path nodes are refreshed (see Fig. 7a). Note that in contrast to the trivial protocol, each remove or add operation only requires an update of a logarithmic (in the size of the group) number of ciphertexts.

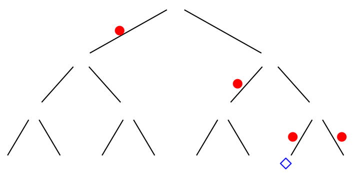
5.1 Pebbling for LKH

Similarly to the case of GGM, the graphs that occur in the security game of LKH are trees. But now, we are interested in randomizing the key at the root of the binary tree structure. This root key can be derived from any of the leaf keys (and publicly available ciphertexts). Hence, there are now many paths to the root which we need to take into account in order to randomize the root key. We therefore will build upon the intricate "edge pebbling" strategy of [Jaf+17] to randomize k_ε, only with guesses replaced by rewindings.

Edge pebbling is a multi-round game on a graph—in our case a binary tree—, where in each step a pebble can be placed on or removed from an edge. The goal



(a) Adding user 4 to a group of 3 users. The keys and ciphertexts that got refreshed in this process are denoted in blue.



(b) A depth-3 binary tree with red pebbles (●) at the edges c_0, c_{10}, c_{110}, and c_{111}. This configuration occurs when pebbling this graph with the pebbling algorithm (see Algorithm 14) at some step i^*. The node leaf_{3,i^*} is marked with a blue diamond (◇).

Fig. 7: (a) Adding a user in LKH. (b) A pebbling configuration occuring in the recursive pebbling strategy from Algorithm 14.

is to "pebble" the tree, which means to reach a pebbling configuration where all edges incident on the root are pebbled. The rule is the following.

Edge-pebbling rule. We can at any point add or remove a pebble on an edge c_x when all of k_x's incoming edges (i.e., edges $c_{x\|0}$ and $c_{x\|1}$, if exist) are pebbled.

In particular, we can pebble or unpebble leaf edges at any point. It is easy to see that we can pebble a binary tree in $2^{d+1} - 2$ steps (by pebbling all edges of the tree, level by level from the leaves to the root). Aiming to reduce the

Algorithm 14: A recursive pebbling algorithm. The "unpebbling" steps use that all pebbling steps are reversible.

```
   Input: A depth-$d$ binary graph with nodes $k_x$ ($x \in \{0,1\}^{\leq d}$) and edges $c_x$
          ($1 \leq |x| \leq d$)
 1 if $d = 1$ then
 2     Pebble $c_0$ and $c_1$                                  // $k_0$ and $k_1$ are leaves
 3 else
 4     Recursively pebble the subgraph rooted at $k_0$        // Pebbles $c_{00}$ and $c_{01}$
 5     Pebble $c_0$                                     // Incoming edges of $k_0$ pebbled
 6     Recursively unpebble subgraph rooted at $k_0$
 7     Recursively pebble the subgraph rooted at $k_1$        // Pebbles $c_{10}$ and $c_{11}$
 8     Pebble $c_1$                                     // Incoming edges of $k_1$ pebbled
 9     Recursively unpebble subgraph rooted at $k_1$
10 end
```

number of pebbles (i.e., the maximum number of pebbled edges at any given point in time), one can observe that a binary tree of depth d can be pebbled in $\Theta(2^{2d})$ steps with only $2d$ pebbles, essentially by a straightforward recursion and removing all used pebbles after pebbling upwards (see Algorithm 14, adapted from [Jaf+17, Algorithm 5]). While for our approach the number of pebbles is not that relevant, this recursive strategy will nevertheless turn out useful. In the following we will derive some useful properties of this strategy.

Definition 9 (Pebbling time). *For $d \in \mathbb{N}$, let T_d be the pebbling time for depth-d binary trees, i.e., the runtime (measured in the number of times a basic pebbling rule is applied) of the pebbling algorithm in Algorithm 14 on a depth-d binary tree.*

Lemma 7. *We have $T_d = (2/3) \cdot (2^{2d} - 1)$.*

Proof. $T_{d+1} = 4T_d + 2$ and $T_1 = 2$ follow immediately from the structure of pebbling algorithm in Algorithm 14 . The claimed closed form of T_d can then be proven, e.g., by induction. □

Definition 10 (Edge index set). *For a given run of pebbling algorithm in Algorithm 14 on a depth-d binary tree as above, let $\mathsf{edges}_{d,i}$ denote the set of indices x of edges c_x pebbled after the i-th step (i.e., application of a pebbling rule).*

Hence, $\mathsf{edges}_{d,0} = \emptyset$ and $\mathsf{edges}_{d,T_d} = \{0,1\}$. A related observation to the following was already used in [Jaf+17].

Lemma 8. *For each $i \in [T_d]_0$, there is a leaf node k_x (for $x \in \{0,1\}^d$) such that both sets $\mathsf{edges}_{d,i-1}$ and $\mathsf{edges}_{d,i}$ (where we set $\mathsf{edges}_{d,-1} := \emptyset$) consist only of edge indices on the path from k_x to k_ε, or its co-path. Formally, for each i, there is an $x \in \{0,1\}^d$, such that for each $x' \| b \in \mathsf{edges}_{d,i-1} \cup \mathsf{edges}_{d,i}$ (for $x' \in \{0,1\}^{<d}$ and $b \in \{0,1\}$), we have $x' \leq_{\mathsf{pfx}} x$.*

Proof. For $d = 1$, this is obvious, as all edges in the tree are incident to the path from the leftmost leaf to the root.

Assume the statement holds for some fixed $d \geq 1$. We will show it holds for $d+1$. Note that the edges c_0 and c_1 lie on the path or co-path of any leaf node. As the subtrees are pebbled or unpebbled recursively as a whole, there are no pebbles in the subtree at n_1 during the pebbling or unpebbling of n_0 and vice versa. Thus, for any of the subtrees, the edge sets at any point consist of the edge set of a subtree of depth d united with potentially the edges c_0 or c_1 which lie on the path or co-path of any leaf. Therefore, the statement also holds for $d+1$. □

Definition 11. *In the situation of Lemma 8, let $\mathsf{leaf}_{d,i}$ be the lexicographically smallest such $x \in \{0,1\}^d$.*

The following corollary is an immediate consequence of Lemma 8.

Corollary 2. *For every i, we have $|\mathsf{edges}_{d,i}| \leq 2d$.*

The following result is an easy consequence of the recursive pebbling strategy.

Lemma 9. *For each $i \in [T_d]_0$, let x_i^* be the unique index in the symmetric difference of* $\mathsf{edges}_{d,i-1}$ *and* $\mathsf{edges}_{d,i}$*, i.e.* $\{x_i^*\} := \mathsf{edges}_{d,i-1} \Delta \mathsf{edges}_{d,i}$. *For each* $x' \in \mathsf{edges}_{d,i-1}$*, it holds that* $|x'| \leq |x_i^*| + 1$.

Proof. If Algorithm 14 is currently at a recursive depth such that $d = 1$ (i.e., x_i^* is incident to a leaf node) the statement follows immediately.

Recall that by Lemma 8, all edges in $\mathsf{edges}_{d,i-1}$ and $\mathsf{edges}_{d,i}$ are incident to the path from $\mathsf{leaf}_{d,i}$ to the root. Thus, when x_i^* is being pebbled (or unpebbled), the only other pebbled edges whose label could be longer than $|x_i^*| + 1$ must be in the subtree rooted at the bottom of x_i^*, as the algorithm first pebbles this subtree before pebbling x_i^* (and thus $\mathsf{leaf}_{d,i}$ must lie in this subtree). At the point when x_i^* is pebbled or unpebbled, the only other pebbled edges in the graph are thus incident to the path from x_i^* to the root (these edges have a label length $|x'| \leq |x_i^*|$), plus the two edges incident to x_i^* directly below x_i^* (these edges have $|x'| = |x_i^*| + 1$). □

Lemmas 8 and 9 immediately imply the following corollary.

Corollary 3. *Let x_i^* be as in Lemma 9. For each $i \in [T_d]_0$, it holds that all edges in* $\mathsf{edges}_{d,i-1} \cup \mathsf{edges}_{d,i}$ *are incident on the unique path from x_i^* to the root.*

As an example, Fig. 7b depicts a state that occurs when pebbling a depth-3 binary tree with Algorithm 14. The set of pebbled edges at this point is $\mathsf{edges}_{3,i} = \{0, 10, 110, 111\}$ where edge c_{111} was pebbled in step i (i.e., $x_i^* = 111$), and $\mathsf{leaf}_{d,i} = 110$.

5.2 A technical lemma

In the following we introduce a technical lemma that will help us in proving closeness of some of the hybrid games for LKH security. In particular, we will be mixing two sampling algorithms, in each of which a bad event can occur. (Later, this bad event will correspond to exceeding a certain bound for repetitions of a loop.) We want to bound the probability for this bad event in a "hybrid" sampling algorithm that starts out as one of the algorithms, and then switches to the other when a specific event occurs.

Lemma 10. *Let $I1, I2, R1, R2$ be randomized algorithms. Let G be a function with the following properties:*

1. *for any sequence $X_0, \dots X_\mathcal{T}$ s.t. $X_0 \leftarrow I1$, $X_t \leftarrow R1(X_{t-1})$ for $t = 1, \dots, \mathcal{T}$, there exists exactly one t such that $G(X_t) = 1$.*
2. *for any sequence $X_0, \dots X_\mathcal{T}$ s.t. $X_0 \leftarrow I2$, $X_t \leftarrow R2(X_{t-1})$ for $t = 1, \dots, \mathcal{T}$, there exists exactly one t such that $G(X_t) = 1$.*
3. *for any index $i = 0, \dots, \mathcal{T}$ it holds that*

$$\Pr_{\substack{X_0 \leftarrow I1 \\ \forall t=1,\dots\mathcal{T}:\ X_t \leftarrow R1(X_{t-1})}} [G(X_i) = 1] = \Pr_{\substack{X_0 \leftarrow I2 \\ \forall t=1,\dots\mathcal{T}:\ X_t \leftarrow R2(X_{t-1})}} [G(X_i) = 1]$$

Algorithm 15: Algorithm P1

```
1 $X_0 \leftarrow I1()$
2 for $t := 1$ to $\mathcal{T}$ do
3     $X_t \leftarrow R1(X_{t-1})$
4 end
```

Algorithm 16: Algorithm P2

```
1 $X_0 \leftarrow I2()$
2 for $t := 1$ to $\mathcal{T}$ do
3     $X_t \leftarrow R2(X_{t-1})$
4 end
```

Algorithm 17: Algorithm P3

```
1 $X_0 \leftarrow I1()$
2 $t = 0$
3 while $\neg G(X_t)$ do
4     $t++$
5     $X_t \leftarrow R1(X_{t-1})$
6 end
7 while $t < \mathcal{T}$ do
8     $t++$
9     $X_t \leftarrow R2(X_{t-1})$
10 end
```

Fig. 8: Algorithms for Lemma 10

4. for any index $i = 0, \dots, \mathcal{T}$, the following identity of distributions holds:

$$\left\{ X_i \,\middle|\, \begin{array}{c} X_0 \leftarrow I1 \\ \forall t \in [\mathcal{T}]\colon X_t \leftarrow R1(X_{t-1}) \\ G(X_i) = 1 \end{array} \right\} \equiv \left\{ X_i \,\middle|\, \begin{array}{c} X_0 \leftarrow I2 \\ \forall t \in [\mathcal{T}]\colon X_t \leftarrow R2(X_{t-1}) \\ G(X_i) = 1 \end{array} \right\}$$

Now consider the algorithms $P1$, $P2$, and $P3$ from Algorithms 15 to 17, and let F be an event that can occur during the sampling processes I1, I2, R1, R2 such that $\Pr[F$ *occurs in* $P1] \leq \gamma_1$ *and* $\Pr[F$ *occurs in* $P2] \leq \gamma_2$.

Then, we have $\Pr[F$ *occurs in* $P3] \leq \gamma_1 + \gamma_2$.

Proof. We start with a technical claim:

Proposition 5. *There exists exactly one i during any run of $P3$ such that $G(X_i) = 1$.*

Proof of Proposition 5. From Item 1 of the lemma hypothesis, we know that during $P3$, G will occur at some point, as $P3$ samples the X_t exactly like $P1$ up to when G occurs.

Due to Item 4, we know that when $G(X_{t^*})$ holds in either $P1$ or $P2$, the states X_{t^*} are identically distributed. As the sampling procedure for $P3$ uses $R2$ (like $P2$ does as well), and $R2$ only takes the previous state as input the "second part" $(X_{t^*+1}, \dots, X_{\mathcal{T}})$ of a state sequence $(X_0, \dots, X_{t^*}, X_{t^*+1}, \dots, X_{\mathcal{T}})$ (where t^* is the first index where $G(X_{t^*}) = 1$) will be identically distributed to $(X'_{t^*+1}, \dots, X'_{\mathcal{T}})$ where $(X'_0, \dots X'_{t^*}, X'_{t^*+1}, \dots, X_{\mathcal{T}})$ is a state sequence generated by $P2$ with $G(X_{t^*})$.

Therefore, $G(X_t) \neq 1$ for all $t > t^*$ due to Item 2. □

Let t^* be the value of t when G occurs for the first time (in any of the three algorithms). We can split up the probability for F as follows:

$$\Pr[F \text{ occurs in } P3] = \Pr[F \text{ occurs in } P3 \text{ up to } t^*] + \Pr[F \text{ occurs in } P3 \text{ after } t^*]$$

$$\leq \Pr_{\substack{X_0 \leftarrow I1 \\ \forall t=1,\ldots\mathcal{T}:\ X_t \leftarrow R1(X_{t-1})}}[F] + \Pr[F \text{ occurs in } P3 \text{ after } t^*]$$

where by "F occurs in $P3$ up to t^*", we denote that F occurs before X_{t^*} is sampled or during the sampling process of X_{t^*} and by "F occurs in $P3$ after t^*" we denote that F occurs during the sampling processes of $X_{t^*+1}, \ldots, X_{\mathcal{T}}$. We now want to bound $\Pr[F \text{ occurs in } P3 \text{ after } t^*]$.

We will write $\Pr[E \text{ occurs in } Pi]$ for $i = 1, 2, 3$ to denote

$$\Pr\left[E \,\middle|\, \begin{array}{c} X_0 \leftarrow I1 \\ \forall t \in [\mathcal{T}]\colon X_t \leftarrow R1(X_{t-1}) \end{array}\right],$$

$$\Pr\left[E \,\middle|\, \begin{array}{c} X_0 \leftarrow I2 \\ \forall t \in [\mathcal{T}]\colon X_t \leftarrow R2(X_{t-1}) \end{array}\right],$$

and

$$\Pr\left[E \,\middle|\, \begin{array}{c} X_0 \leftarrow I1 \\ \forall t = 1, \ldots t^*\colon X_t \leftarrow R1(X_{t-1}) \\ \forall t = t^*+1, \ldots \mathcal{T}\colon X_t = R2(t, X_{t-1}) \end{array}\right],$$

respectively, i.e., the probability of E happening when the sampling of the states X_i is done according to the processes $P1, P2$, or $P3$, respectively.

$$\begin{aligned}
&\Pr[F \text{ occurs after } t^* \text{ in } P3] \\
\overset{\text{Proposition 5}}{=}\ & \sum_{t=0}^{\mathcal{T}} \Pr[F \text{ occurs after } t^* \text{ in } P3 \wedge t^* = t \text{ in } P3] \\
=\ & \sum_{t=0}^{\mathcal{T}} \Pr[F \text{ occurs after } t^* \text{ in } P3 \mid t^* = t \text{ in } P3] \cdot \Pr[t^* = t \text{ in } P3] \\
=\ & \sum_{t=0}^{\mathcal{T}} \Pr[F \text{ occurs after } t^* \text{ in } P3 \mid t^* = t \text{ in } P3] \cdot \Pr[t^* = t \text{ in } P1] \\
\overset{\text{Item 3}}{=}\ & \sum_{t=0}^{\mathcal{T}} \Pr[F \text{ occurs after } t^* \text{ in } P3 \mid t^* = t \text{ in } P3] \cdot \Pr[t^* = t \text{ in } P2] \\
\overset{\text{Item 4}}{=}\ & \sum_{t=0}^{\mathcal{T}} \Pr[F \text{ occurs after } t^* \text{ in } P2 \mid t^* = t \text{ in } P2] \cdot \Pr[t^* = t \text{ in } P2] \\
=\ & \sum_{t=0}^{\mathcal{T}} \Pr[F \text{ occurs after } t^* \text{ in } P2 \wedge t^* = t \text{ in } P2] \\
\leq\ & \Pr[F \text{ occurs in } P2] \leq \gamma_2
\end{aligned}$$

Altogether, this yields

$$\Pr[F \text{ occurs in } P3] \ \leq\ \gamma_1 + \gamma_2.$$

□

5.3 Proving security from IND-CPA

We now define the LKH security experiment $\mathrm{Exp}^{\mathsf{LKH}}_{\mathsf{SKE},\mathcal{A},d}$. This game models the security of LKH as an MKD protocol.

Definition 12 (LKH security experiment). *Let* $\mathsf{SKE} = (\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec})$ *be a secret-key encryption scheme and* $d = d(\lambda)$ *some depth. The* LKH *experiment* $\mathrm{Exp}^{\mathsf{LKH}}_{\mathsf{SKE},\mathcal{A},d}$ *initially samples uniformly at random a challenge bit* $b_{\mathsf{lkh}} \leftarrow \{0,1\}$ *and keys* $k_x \leftarrow \{0,1\}^\lambda$ *for each* $x \in \{0,1\}^{\leq d}$*. It then computes ciphertexts* $c_{x\|b}$ *for all* $x \in \{0,1\}^{<d}$ *and* $b \in \{0,1\}$*, which encrypt key* k_x *under key* $k_{x\|b}$*. The adversary* $\mathcal{A}$ *receives the ciphertexts* $c_{x\|b}$ *and can then make the following queries:*

– *$\mathcal{A}$ can adaptively corrupt "leaf" keys k_x (for $x \in \{0,1\}^d$). This exposes k_x to $\mathcal{A}$, and results in a refresh of not only k_x, but also all $k_{x'}$ for proper prefixes x' of x. Furthermore, fresh encryptions of those $k_{x'}$ under keys $k_{x'\|0}$ and $k_{x'\|1}$ are generated and exposed to $\mathcal{A}$.*

– *At any point, $\mathcal{A}$ may stop the game by asking to be challenged to distinguish the then-current key k_ε (case $b_{\mathsf{lkh}} = 0$) from a random key (case $b_{\mathsf{lkh}} = 1$).*

The output of the experiment is 1 if $\mathcal{A}$ correctly guesses the bit b_{lkh}, and 0 otherwise. We define the advantage of $\mathcal{A}$ in this game as

$$\mathrm{Adv}^{\mathsf{LKH}}_{\mathsf{SKE},\mathcal{A},d}(\lambda) := \Pr[\mathrm{Exp}^{\mathsf{LKH}}_{\mathsf{SKE},\mathcal{A},d}(\lambda) = 1] - 1/2.$$

Asymptotically, we say that LKH *is secure (with* SKE*) if for every polynomial-time $\mathcal{A}$ and every constant $c \in \mathbb{N}$, the advantage $\mathrm{Adv}^{\mathsf{LKH}}_{\mathsf{SKE},\mathcal{A},c\cdot\log(\lambda)}(\lambda)$ is negligible.*[11]

Our strategy. We will prove that $\mathcal{A}$ has a negligible advantage to win the game $\mathrm{Exp}^{\mathsf{LKH}}_{\mathsf{SKE},\mathcal{A},d}$ through a large hybrid argument. We will start with the game above and apply a number of successive changes until finally $\mathcal{A}$'s view is independent of the real final key k_ε. Similar to Section 4, we make a liberal use of rewindings, thus, it will be helpful to formalize $\mathcal{A}$'s view:

Definition 13 (Adversarial view). *In a run of the* LKH *experiment* $\mathrm{Exp}^{\mathsf{LKH}}_{\mathsf{SKE},\mathcal{A},d}$ *from Definition 12, we define $\mathcal{A}$'s view* $\mathsf{view}_\mathcal{A}$ *in this run as a sequence* $(\mathsf{ev}_1, \ldots, \mathsf{ev}_\mathcal{T})$ *of events, where each* ev_i *can be one of the following:*

Query. *One of $\mathcal{A}$'s queries (without reply), either of the form* $(\mathsf{corrupt}, x)$*, or* $\mathsf{challenge}$.

New key. *Every time a new key k_x is defined, right before that, a corresponding* (newkey, x) *event is appended to* view*. Concretely,* view *starts with* (newkey, x) *events for* $x \in \{0,1\}^{\leq d}$*. Furthermore, a query* $(\mathsf{corrupt}, x)$ *automatically causes also entries* (newkey, x') *for a prefix x' of x to be appended immediately after that* $\mathsf{corrupt}$ *entry. Entries* (newkey, x) *defined at the same time are ordered in* view *with keys further from the root (i.e., with longer x) first.*

[11]Like previous works, we focus on a logarithmic depth and thus to polynomially many users.

Ciphertext. *An event* $(\mathsf{ctxt}, x\|b, c_{x\|b})$ *for a ciphertext* $c_{x\|b} = \mathbf{Enc}(k_{x\|b}, k_x)$, *either as part of* $\mathcal{A}$*'s initial input, or as a side effect of a corruption query.* ctxt *entries defined at the same time are ordered with ciphertexts furthest from the root (i.e., with longer* x*) first, and lexicographically (according to* $x\|b$*) for* x *of the same length.*

Corrupted key. *A key* (key, x, k_x) *as a result of a corruption query. We assume that this key appears before the corresponding new key events and the ciphertexts that are sent to* $\mathcal{A}$ *in the same reply.*

Challenge key. *The result of the final challenge query, in the form* $(\mathsf{challenge}, k)$ *(i.e., depending on* $b_{\mathcal{B}}$ *with either* k *being a key* k_ε *or a random value).*

Decision bit. *The final output bit* $b_{\mathcal{A}}$ *of* $\mathcal{A}$*, in the form* $(\mathsf{guess}, b_{\mathcal{A}})$*. This event is the last in* view*, and we may write* $\mathbf{out}_{\mathcal{A}}(\mathsf{view})$ *for the output bit* $b_{\mathcal{A}}$*.*

Remark 2. Note that the ordering of events above implies for an event $(\mathsf{corrupt}, x)$ that it is followed by an event (key, x, k_x), then a sequence of events (newkey, x') for all prefixes x' of x, in decreasing length, and then a sequence $(\mathsf{ctxt}, x'\|b, c_{x'\|b})$ for all strict prefixes x' of x and bits $b \in \{0,1\}$, again ordered by decreasing length, and siblings ordered alphabetically. For example, for depth $d = 3$, a $(\mathsf{corrupt}, 010)$ event causes the following sequence of events: $(\mathsf{key}, 010, k_{010})$, $(\mathsf{newkey}, 010)$, $(\mathsf{newkey}, 01)$, $(\mathsf{newkey}, 0)$, $(\mathsf{newkey}, \varepsilon)$, $(\mathsf{ctxt}, 010, c_{010})$, $(\mathsf{ctxt}, 011, c_{011})$, $(\mathsf{ctxt}, 00, c_{00})$, $(\mathsf{ctxt}, 01, c_{01})$, $(\mathsf{ctxt}, 0, c_0)$, and $(\mathsf{ctxt}, 1, c_1)$.

We are now ready to formulate and prove our main result:

Theorem 2. *Let* $\mathsf{SKE} = (\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec})$ *be an SKE scheme. Then*
– *for every* LKH *adversary* $\mathcal{A}$ *that runs in time* $t_{\mathcal{A}}$ *and places at most* $\mathcal{Q}_{\mathsf{corrupt}}$ *corruption queries,*
– *for every* LKH *depth* d *and every* $\gamma \in (0,1]$,

there is an $\mathsf{IND\text{-}CPA}$ *adversary* $\mathcal{B}$ *that makes at most* $\mathcal{Q}_{\mathsf{LoR}} \leq 2 \cdot \ln(2\mathcal{T}/\gamma) \cdot \mathcal{T}^4 \cdot (d+1)/\gamma$ *LoR queries,* $\mathcal{Q}_{\mathsf{NU}} \leq 2 \cdot \ln(2\mathcal{T}/\gamma) \cdot \mathcal{T}^3 \cdot (d+1)/\gamma$ *new user queries, and runs in time* $t_{\mathcal{B}}$ *and for which*

$$\mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE},\mathcal{B}}(\lambda) \geq \frac{1}{2} \cdot \frac{1}{T_d} \cdot \mathrm{Adv}^{\mathsf{LKH}}_{\mathsf{SKE},\mathcal{A},d}(\lambda) - \frac{\gamma}{2}. \tag{12}$$

where

$$t_{\mathcal{B}} \ \lessapprox\ 2 \cdot \ln(2\mathcal{T}/\gamma) \cdot \mathcal{T}^4 \cdot (d+1)/\gamma \cdot t_{\mathcal{A}}. \tag{13}$$

where $\mathcal{T} \leq 2^{d+2} + (3d+1) \cdot \mathcal{Q}_{\mathsf{corrupt}}$.

Again, before proceeding to a proof, we remark that Theorem 2 implies asymptotic security when setting γ suitably:

Corollary 4 (SKE IND-CPA $\Rightarrow$ LKH secure). *If* SKE *is* $\mathsf{IND\text{-}CPA}$ *secure (as in Definition 5), then* LKH *is secure with* SKE *(in the sense of Definition 12).*

Proof of Corollary 4. Fix a depth $d = c \cdot \log(\lambda)$ (for some constant $c \in \mathbb{N}$), and assume for contradiction that there is a polynomial-time adversary $\mathcal{A}$ against the security of LKH with non-negligible advantage $\varepsilon_{\mathcal{A}} := \mathrm{Adv}^{\mathsf{LKH}}_{\mathsf{SKE},\mathcal{A},d}(\lambda)$. Since

Algorithm 18: Game i, with the bulk of the work outsourced into the sampling from $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}$.

```
1 $b_{\mathsf{lkh}} \leftarrow \{0,1\}$
2 $\mathsf{view} \leftarrow \mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}$          // view has the format from Definition 13
3 return $[b_{\mathsf{lkh}} = \mathbf{out}_{\mathcal{A}}(\mathsf{view})]$          // returns 1 iff $b_{\mathsf{lkh}} = \mathbf{out}_{\mathcal{A}}(\mathsf{view})$
```

$\varepsilon_{\mathcal{A}}$ is non-negligible, there exists a polynomial p such that for infinitely many values of λ, we have $\varepsilon_{\mathcal{A}} \geq 1/p(\lambda)$.

Now set $\gamma = 1/(2T_d p(\lambda))$ (for the value T_d from Definition 9, which is polynomially bounded by our choice of d and by Lemma 7), and invoke Theorem 2. We obtain an $\mathsf{IND\text{-}CPA}$ adversary $\mathcal{B}$ with (by (13)) polynomial runtime and non-negligible advantage

$$\mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE},\mathcal{B}}(\lambda) \overset{(12)}{\geq} \frac{1}{2T_d}\cdot\varepsilon_{\mathcal{A}} - \frac{\gamma}{2} \overset{(*)}{\geq} \frac{1}{2T_d}\cdot\frac{1}{p(\lambda)} - \frac{\gamma}{2} = \frac{1}{4T_d p(\lambda)},$$

where $(*)$ holds (only) for infinitely many λ. □

Proof Overview. Fix SKE, $\mathcal{A}$, and d. In the following, we will consider a number of hybrid games, with Game lkh being the original LKH experiment. Denoting with $\mathbf{out}_i$ the output of Game i, we trivially get

$$\Pr[\mathbf{out}_{\mathsf{lkh}} = 1] = \mathrm{Adv}^{\mathsf{LKH}}_{\mathsf{SKE},\mathcal{A},d}(\lambda) + 1/2. \tag{14}$$

To move on, we will formulate Game i (for $0 \leq i \leq T_d$) in a (for us) convenient way, see Algorithm 18. This formulation outsources the bulk of the game into the sampling of $\mathcal{A}$'s view view from a suitable distribution $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}$. In our upcoming refinements, we will only change $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}$ and investigate the effects on $\mathbf{out}_i$.

The distributions $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}$. To define the distribution $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}$ for Game i, we use the following notation:

- $\mathcal{D}^{\mathsf{view}}_{\mathsf{lkh},b_{\mathsf{lkh}}}$ is the distribution of $\mathcal{A}$-views (as in Definition 13) that is induced by running the LKH experiment with challenge bit b_{lkh} (that decides whether $\mathcal{A}$ is challenged with k_ε or a random key).
- $\mathsf{edges}_{d,i}$ is the edge index set from Definition 10 that arises out of pebbling a depth-d binary tree.
- $\mathsf{len}(\mathsf{view})$ is the length of a given $\mathcal{A}$-view view (measured in events).
- $\mathsf{pfx}_t(\mathsf{view})$ outputs the prefix of view up to (and including) the t-th event (see Section 2.1).
- $\mathsf{maxcor}_{i,t}(\mathsf{view})$ on input $\mathsf{view} = (\mathsf{ev}_1,\ldots,\mathsf{ev}_{\mathcal{T}})$ outputs
$$\max\left(\{\, |x| \mid \mathsf{ev}_{t'} = (\mathsf{ctxt}, x, c_x) \text{ for some } t' > t \text{ and } x \leq_{\mathsf{pfx}} \mathsf{leaf}_{d,i} \,\} \cup \{0\}\right).$$
- $\mathsf{lastkey}_i(\mathsf{view})$ on input $\mathsf{view} = (\mathsf{ev}_1,\ldots,\mathsf{ev}_{\mathcal{T}})$ outputs
$$\max\{\, t' \mid \mathsf{ev}_{t'} = (\mathsf{newkey}, x_i^*)\},$$
where $x_i^* := \mathsf{edges}_{d,i-1} \Delta \mathsf{edges}_{d,i}$ for $i \geq 1$ and $x_0^* := \mathsf{edges}_{d,0} \Delta \mathsf{edges}_{d,1}$.

– $B \in \mathbb{N}$ is a bound on the number of repetitions of Lines 6 to 14 for each t. In case of B unsuccessful repetitions for one t, the whole algorithm outputs $\perp$. We will fix a suitable value for B later.

Below we will prove some useful properties of the functions maxcor and $\mathsf{lastkey}$, and are now ready to define the distributions $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}$; see Algorithm 19. Our distribution $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}$ is defined like $\mathcal{D}^{\mathsf{view}}_{\mathsf{lkh},b_{\mathsf{lkh}}}$ (i.e., like an LKH run with $\mathcal{A}$), but uses rewinding and resampling (as defined in Section 4) at every step. Additionally, we replace certain ciphertexts c_x as indicated by $x \in \mathsf{edges}_{d,i}$ during the rewindings. Concretely, fix an i and consider Algorithm 19, which programmatically describes sampling $\mathsf{view}_{\mathcal{T}}$ according to $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}$.

Algorithm 19: Sampler for $\mathcal{D}^{\mathsf{view}}_{i\boxed{.1},b_{\mathsf{lkh}}}$

```
Input: $i \in \{0,\dots,T_d\}, b_{\mathsf{lkh}} \in \{0,1\}, B \in \mathbb{N}$   // $\mathsf{len}, \mathsf{pfx}_t, \mathsf{maxcor}_{i,t}, \mathsf{lastkey}_i, B$ described in proof
1  $\mathsf{view}_0 \leftarrow \mathcal{D}^{\mathsf{view}}_{\mathsf{lkh},b_{\mathsf{lkh}}}$
2  $\mathcal{T} := \mathsf{len}(\mathsf{view}_{\mathsf{lkh}})$   // Length of $\mathsf{view}_{\mathsf{lkh}}$ (in entries)
3  for $t := 1$ to $\mathcal{T}$ do
4      Write $\mathsf{view}_{t-1} = (\mathsf{ev}_{t-1,1},\dots,\mathsf{ev}_{t-1,\mathcal{T}})$
5      repeat   // Output $\perp$ if $B$ repetitions fail for this $t$
6          Rewind adversary to point $t$   // Checks if $\mathsf{ev}_{t,t}$ defines a ciphertext to be pebbled
7          if $\mathsf{ev}_{t-1,t} = (\mathsf{ctxt}, x, c_x)$ with $x \in \mathsf{edges}_{d,i}$ and $\mathsf{maxcor}_{i\boxed{+1},t+1}(\mathsf{view}_{t-1}) < |x|$ then
8              Sample fresh $c_x^{\perp} \leftarrow \mathbf{Enc}(k_x, \perp)$   // Fresh dummy ciphertext
9              Set $\mathsf{ev}_{t,t} := (\mathsf{ctxt}, x, c_x^{\perp})$ in $\mathsf{view}_t$   // Replace $c_x$ with $c_x^{\perp}$ in $\mathsf{view}_{t-1}$
10             Resample from point $t+1$ to obtain $\mathsf{view}_t = (\mathsf{ev}_{t-1,0},\dots,\mathsf{ev}_{t-1,t-1},\mathsf{ev}_{t,t},\mathsf{ev}_{t,t+1},\dots,\mathsf{ev}_{t,\tau})$
11         else
12             Resample from point $t$ to obtain $\mathsf{view}_t = (\mathsf{ev}_{t-1,0},\dots,\mathsf{ev}_{t-1,t-1},\mathsf{ev}_{t,t},\dots,\mathsf{ev}_{t,\tau})$
13         end
14     until $\mathsf{maxcor}_{i\boxed{+1},t+1}(\mathsf{view}_t) = \mathsf{maxcor}_{i\boxed{+1},t+1}(\mathsf{view}_{t-1})$ and $\mathsf{len}(\mathsf{view}_t) = \mathsf{len}(\mathsf{view}_{t-1})$ and $\mathsf{lastkey}_{i\boxed{+1}}(\mathsf{view}_t) = \mathsf{lastkey}_{i\boxed{+1}}(\mathsf{view}_{t-1})$
15 end
16 return $\mathsf{view}_{\mathcal{T}}$
```

Having defined our hybrid distributions $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}$, we will additionally consider potentially inefficient procedures $\widetilde{\mathcal{D}}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}$ which are defined similar to Algorithm 19 but without a bound B on the runtime. We will first show that the distribution $\widetilde{\mathcal{D}}^{\mathsf{view}}_{0,b_{\mathsf{lkh}}}$ coincides with the distribution of views in Game lkh. Next, we will consider the intermediate distributions $\mathcal{D}^{\mathsf{view}}_{i.1,b_{\mathsf{lkh}}}$ (the difference to $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}$ being marked gray in Algorithm 19) and show that for all $i \in [T_d - 1]_0$ it holds that $\widetilde{\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}}$ and $\widetilde{\mathcal{D}^{\mathsf{view}}_{i.1,b_{\mathsf{lkh}}}}$ have the same distribution.

We will then bound the difference in the probability of an abort in the games $\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}$ and $\mathcal{D}^{\mathsf{view}}_{i.1,b_{\mathsf{lkh}}}$. To prove this we will need an additional technical lemma about the abort probability in such mixed sampling procedures and this is the main difference to Section 4 in which such a mixed resampling procedure does not occur. Setting the bound B appropriately we will be able to bound the probability of an abort in $\mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{lkh}}}$ by γ. In the subsequent claim we will show that $\mathcal{A}$ has no advantage in Game T_d since the view sampled according to $\mathcal{D}^{\mathsf{view}}_{T_d,b_{\mathsf{lkh}}}$ is independent of b_{lkh}. Finally, we will conclude the proof by arguing that for all $i \in [T_d - 1]_0$ the distributions $\mathcal{D}^{\mathsf{view}}_{i.1,b_{\mathsf{lkh}}}$ and $\mathcal{D}^{\mathsf{view}}_{i+1,b_{\mathsf{lkh}}}$ are computationally indistinguishable by $\mathsf{IND\text{-}CPA}$ security of the SKE scheme SKE. See Fig. 9 for an overview of this sequence of arguments.

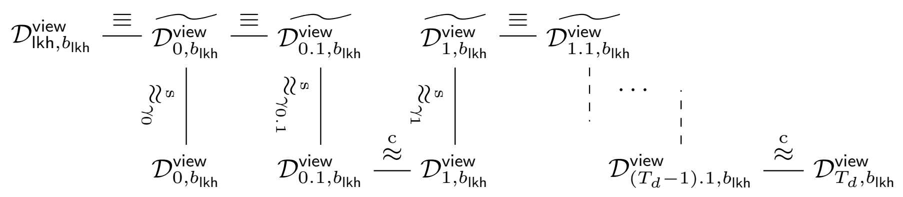

$$\begin{tikzcd}
\mathcal{D}^{\mathsf{view}}_{\mathsf{lkh},b_{\mathsf{lkh}}} \arrow[r, dash, "\equiv"] & \widetilde{\mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{lkh}}}} \arrow[r, dash, "\equiv"] \arrow[d, dash, "\overset{s}{\approx}_{\gamma_0}"'] & \widetilde{\mathcal{D}^{\mathsf{view}}_{0.1,b_{\mathsf{lkh}}}} \arrow[d, dash, "\overset{s}{\approx}_{\gamma_{0.1}}"'] & \widetilde{\mathcal{D}^{\mathsf{view}}_{1,b_{\mathsf{lkh}}}} \arrow[r, dash, "\equiv"] \arrow[d, dash, "\overset{s}{\approx}_{\gamma_1}"'] & \widetilde{\mathcal{D}^{\mathsf{view}}_{1.1,b_{\mathsf{lkh}}}} \arrow[d, dashed, dash] & \cdots & & \\
& \mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{lkh}}} & \mathcal{D}^{\mathsf{view}}_{0.1,b_{\mathsf{lkh}}} \arrow[r, dash, "\overset{c}{\approx}"] & \mathcal{D}^{\mathsf{view}}_{1,b_{\mathsf{lkh}}} & & & \mathcal{D}^{\mathsf{view}}_{(T_d-1).1,b_{\mathsf{lkh}}} \arrow[r, dash, "\overset{c}{\approx}"] & \mathcal{D}^{\mathsf{view}}_{T_d,b_{\mathsf{lkh}}}
\end{tikzcd}$$

Fig. 9: Sequence of hybrids from the proof of Theorem 2. Perfect indistinguishabilities ("$\equiv$") are shown in Proposition 6 and Proposition 7, statistical indistinguishabilities ("$\overset{s}{\approx}_{\gamma_i}$" and "$\overset{s}{\approx}_{\gamma_{i.1}}$") with bounds γ_i and $\gamma_{i.1}$ follow from our $\mathsf{IND\text{-}CPA}$ reduction of Theorem 2 and Proposition 8, and computational indistinguishabilities ("$\overset{c}{\approx}$") follow from the $\mathsf{IND\text{-}CPA}$ reduction. Note that the statement $\widetilde{\mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{lkh}}}} \overset{s}{\approx}_{\gamma_0} \mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{lkh}}}$ is needed to bound $\gamma_{0.1}$.

Some useful properties of maxcor and $\mathsf{lastkey}$. We prove some useful properties of maxcor and $\mathsf{lastkey}$.

Lemma 11. *For any $i \in \{0, \ldots, T_d\}$, any view as described above, and any $t \in \{1, \ldots, \mathsf{len}(\mathsf{view}) - 1\}$, it holds that $\mathsf{maxcor}_{i,t}(\mathsf{view}) \geq \mathsf{maxcor}_{i,t+1}(\mathsf{view})$.*

Proof. This follows from the fact that maxcor considers the suffix of view and so increasing from t to $t+1$ only removes events that are considered for maxcor. □

We show that for edges that we want to put pebbles on, it does not matter whether we look at the maxcor value for i or for $i+1$ to determine when is a good time to put a pebble. This will be useful when game hopping later in the proof.

Lemma 12. *For any* $i \in \{0, \dots, T_d - 1\}$*, any* view *as described above, any* $t \in \{1, \dots, \mathsf{len}(\mathsf{view})\}$*, any* $x \in \mathsf{edges}_{d,i}$*, it holds*

$$(\mathsf{maxcor}_{i,t}(\mathsf{view}) < |x|) \Leftrightarrow (\mathsf{maxcor}_{i+1,t}(\mathsf{view}) < |x|).$$

Proof. Let x'_{i+1} be the longest common prefix of $\mathsf{leaf}_{d,i}$ and $\mathsf{leaf}_{d,i+1}$.

We note that by Lemma 8, as $x \in \mathsf{edges}_{d,i}$, x lies on the path or the co-path of $x'_{i+1}\|0$, so in particular $|x| \leq \left|x'_{i+1}\|0\right|$. This holds because $\mathsf{edges}_{d,i}$ is a subset of both sets of edges incident on the path to $\mathsf{leaf}_{d,i}$ and $\mathsf{leaf}_{d,i+1}$.

If $\mathsf{maxcor}_{i,t+1}(\mathsf{view}_{t-1}) < |x|$, this in particular means that the length-maximal x' fulfilling the criterion from the definition of $\mathsf{maxcor}_{i,t+1}(\mathsf{view}_{t-1})$ must have $|x'| < |x|$. Thus, it must be a prefix of x'_{i+1}. Thus, it is also a prefix of $\mathsf{leaf}_{d,i+1}$. As for each (ctxt, x, c_x) event, the sibling event $(\mathsf{ctxt}, \mathsf{pfx}_{|x|-1}(x)\|(1 - x_{|x|}), c_{\mathsf{pfx}_{|x|-1}(x)\|(1-x_{|x|})})$ must also occur right before or right after, this yields that if x' is the longest prefix of $\mathsf{leaf}_{d,i}$ after $t+1$, it must also be the longest prefix of $\mathsf{leaf}_{d,i+1}$ after $t+1$. This yields one implication of the equivalence. The reverse direction follows by a symmetrical argument. □

The following lemma states that the last corruption of the relevant key will already have happened before a pebble is embedded (i.e. before a ciphertext is replaced by an encryption of $\bot$). This will be useful to see that the end conditions of the **repeat** loops in different versions of Algorithm 19 are equivalent.

Lemma 13. *Let* $i \in [T_d]$ *and* $x \in \mathsf{edges}_{d,i}$*. Then,* $\mathsf{maxcor}_{i,t+1}(\mathsf{view}) < |x|$ *implies that* $\mathsf{lastkey}_i(\mathsf{view}) < t+1$ *and* $\mathsf{lastkey}_{i+1}(\mathsf{view}) < t+1$*.*

Proof. Let x_i^* be as in the definition of $\mathsf{lastkey}_i$. By Corollary 3, any $x \in \mathsf{edges}_{d,i}$ is incident on the path from x_i^* to the root. Thus $|x| \leq |x_i^*| + 1$. It follows that if $\mathsf{maxcor}_{i,t+1}(\mathsf{view}) < |x|$, then also $\mathsf{maxcor}_{i,t+1}(\mathsf{view}) < |x_i^*| + 1$. As any event of the form (newkey, x^*) triggers events of the form $(\mathsf{ctxt}, x^*\|b, c_{x^*\|b})$ for $b \in \{0,1\}$, any (newkey, x_i^*) event at or after $t+1$ implies that $\mathsf{maxcor}_{i,t+1}(\mathsf{view}) \geq |x_i^*| + 1$. The case for $\mathsf{lastkey}_{i+1}$ follows from Lemma 12. □

We apply the above lemmas to views that share prefixes to find that identical maxcor values imply identical $\mathsf{lastkey}$ values as soon as a pebble is embedded. This will again be useful in a game hop.

Corollary 5. *Let* t, i *be arbitrary, and let* $\mathsf{view} = (\mathsf{ev}_1, \dots, \mathsf{ev}_{\mathcal{T}})$ *be such that* $\mathsf{ev}_t = (\mathsf{ctxt}, x, c_x)$ *with* $x \in \mathsf{edges}_{d,i}$ *and* $\mathsf{maxcor}_{i,t+1}(\mathsf{view}) < |x|$*. Then, for any* view' *with* $\mathsf{pfx}_t(\mathsf{view}') = \mathsf{pfx}_t(\mathsf{view})$*, we have*

$$\mathsf{maxcor}_{i,t+1}(\mathsf{view}') = \mathsf{maxcor}_{i,t+1}(\mathsf{view}) \;\Rightarrow\; \mathsf{lastkey}_i(\mathsf{view}') = \mathsf{lastkey}_i(\mathsf{view})$$

and

$$\mathsf{maxcor}_{i+1,t+1}(\mathsf{view}') = \mathsf{maxcor}_{i+1,t+1}(\mathsf{view}) \;\Rightarrow\; \mathsf{lastkey}_{i+1}(\mathsf{view}') = \mathsf{lastkey}_{i+1}(\mathsf{view}).$$

Proof. As $\mathsf{maxcor}_{i,t+1}(\mathsf{view}) < |x|$, by Lemma 12 $\mathsf{maxcor}_{i+1,t+1}(\mathsf{view}) < |x|$. Furthermore, if $\mathsf{maxcor}_{i,t+1}(\mathsf{view}) = \mathsf{maxcor}_{i,t+1}(\mathsf{view}')$, then also $\mathsf{maxcor}_{i,t+1}(\mathsf{view}') < |x|$ and the same implication as above holds for view'. From Lemma 13, it follows that $\mathsf{lastkey}_i(\mathsf{view}) < t+1$ and $\mathsf{lastkey}_i(\mathsf{view}') < t+1$. Therefore, as the prefixes up to t of view and view' are the same, in fact $\mathsf{lastkey}_i(\mathsf{view}') = \mathsf{lastkey}_i(\mathsf{view})$. Using a similar argument, the implication for $i+1$ also follows. □

Proof of Theorem 2. We are now ready to prove Theorem 2.

Proof. We start with a few helper propositions to structure our proof.

Proposition 6. $\widetilde{\mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{lkh}}}} \equiv \mathcal{D}^{\mathsf{view}}_{\mathsf{lkh},b_{\mathsf{lkh}}}$.

Proof of Proposition 6. Since $\mathsf{edges}_{d,0} = \emptyset$, the **if** clause can never return true and the algorithm always enters the **else** clause. The statement therefore follows from Lemma 5, where $\mathcal{D} = \mathcal{D}^{\mathsf{view}}_{\mathsf{lkh},b_{\mathsf{lkh}}}$, $h_t(\mathsf{view}_s) = (\mathsf{ev}_{s,1}, \dots, \mathsf{ev}_{s,t-1})$, and $g_t(\mathsf{view}_s) = (\mathsf{maxcor}_{0,t}(\mathsf{view}_s), \mathsf{len}(\mathsf{view}_s), \mathsf{lastkey}_0(\mathsf{view}_s))$. □

Proposition 7 **($\widetilde{\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}} \equiv \widetilde{\mathcal{D}^{\mathsf{view}}_{i.1,b_{\mathsf{lkh}}}}$).** *For all $i \in [T_d - 1]_0$, we have $\widetilde{\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}} \equiv \widetilde{\mathcal{D}^{\mathsf{view}}_{i.1,b_{\mathsf{lkh}}}}$.*

Proof of Proposition 7. By Lemma 5, when instantiated once with

$$\begin{aligned} h_t(\mathsf{view}) &= \mathsf{pfx}_{t-1}(\mathsf{view}),\\ g_t(\mathsf{view}) &= (\mathsf{maxcor}_{i,t+1}(\mathsf{view}), \mathsf{len}(\mathsf{view}), \mathsf{lastkey}_i(\mathsf{view})), \end{aligned}$$

and once with h_t as above and

$$g_t(\mathsf{view}) = (\mathsf{maxcor}_{i+1,t+1}(\mathsf{view}), \mathsf{len}(\mathsf{view}), \mathsf{lastkey}_{i+1}(\mathsf{view})),$$

the distributions of the view_t up until the **if** in Line 7 of Algorithm 19 returns true for the first time are identical. Further, by Lemma 12, the conditions for the **if** are equivalent, i.e. whenever the **if** condition would return true for $i+1$ it would also return true for i and vice versa. It therefore remains to show that the end conditions of the **repeat** loop are also equivalent after the first time the **if** returned true.

To see this, note that after the first time the **if** returned true, $\mathsf{maxcor}_{i,t+1}$ as well as $\mathsf{maxcor}_{i+1,t+1}$ are upper bounded by the length of any of the edge labels in $\mathsf{edges}_{d,i}$ – this follows from Lemma 12 and Lemma 11.

As by Lemma 9, all of these edges are incident on the path from the longest common prefix of $\mathsf{leaf}_{d,i}$ and $\mathsf{leaf}_{d,i+1}$ to the root or its co-path.

Thus, if $\mathsf{maxcor}_{i,t+1}$ and $\mathsf{maxcor}_{i+1,t+1}$ are bounded by the length of such an edge label, it follows in fact that $\mathsf{maxcor}_{i,t+1}(\mathsf{view}) = \mathsf{maxcor}_{i+1,t+1}(\mathsf{view})$.

By a similar argument, and by Corollary 5, it follows that the part of the stopping condition that concerns $\mathsf{lastkey}$ is equivalent.

Lastly, we see that $\mathsf{len}(\mathsf{view}_t)$ does not depend on i and thus the stopping conditions of the **repeat** loops in the two algorithms are equivalent.

The statement follows. □

Proposition 8. *For all $i \in [T_d - 1]_0$, we have*

$$\gamma_{i.1} = \Pr[\perp \leftarrow \mathcal{D}^{\mathsf{view}}_{i.1,b_{\mathsf{lkh}}}] \;\le\; \Pr[\perp \leftarrow \mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}] + \gamma.$$

Proof of Proposition 8. We use Lemma 10, where event F will be exceeding the bound B on the iterations of the **repeat** loop. We further define the procedures $I1, I2, R1_i, R2_i$ as listed in Algorithms 20 to 22.

Algorithm 20: $I1$ and $I2$

```
1 $\mathsf{view}_0 \leftarrow \mathcal{D}^{\mathsf{view}}_{\mathsf{lkh},b_{\mathsf{lkh}}}$
2 return $X_0 = (0, \mathsf{view}_0)$
```

Algorithm 21: $R1_i$

```
  Input: $(t-1, \mathsf{view}_{t-1})$
1 repeat
2     Rewind adversary to point $t$
3     Resample from point $t$
4 until $\mathsf{maxcor}_{i,t+1}(\mathsf{view}_t) = \mathsf{maxcor}_{i,t+1}(\mathsf{view}_{t-1})$ and
      $\mathsf{len}(\mathsf{view}_t) = \mathsf{len}(\mathsf{view}_{t-1})$ and
      $\mathsf{lastkey}_i(\mathsf{view}_t) = \mathsf{lastkey}_i(\mathsf{view}_{t-1})$
5 return $X_t = (t, \mathsf{view}_t)$
```

Algorithm 22: $R2_i$

```
   Input: $(t-1, \mathsf{view}_{t-1})$
1  repeat
2      Rewind adversary to point $t$
3      if $\mathsf{ev}_{t,t} = (\mathsf{ctxt}, x, c_x)$ with $x \in \mathsf{edges}_{d,i}$ and
         $\mathsf{maxcor}_{i+1,t+1}(\mathsf{view}_{t-1}) < |x|$ then
4          Sample fresh $c_x^{\perp} \leftarrow \mathbf{Enc}(k_x, \perp)$
5          Set $\mathsf{ev}_{t,t} := (\mathsf{ctxt}, x, c_x^{\perp})$ in $\mathsf{view}_t$
6          Resample from point $t+1$
7      else
8          Resample from point $t$
9      end
10 until $\mathsf{maxcor}_{i+1,t+1}(\mathsf{view}_t) = \mathsf{maxcor}_{i+1,t+1}(\mathsf{view}_{t-1})$ and
       $\mathsf{len}(\mathsf{view}_t) = \mathsf{len}(\mathsf{view}_{t-1})$ and
       $\mathsf{lastkey}_{i+1}(\mathsf{view}_t) = \mathsf{lastkey}_{i+1}(\mathsf{view}_{t-1})$
11 return $X_t = (t, \mathsf{view}_t)$
```

Fig. 10: Algorithms for the proof of Proposition 8

In the following, we will want to map the first placement of a pebble in Algorithm 19 to the event G that triggers switching the algorithms in Lemma 10, i.e. if $G(t, \mathsf{view}_t)$ is true, the next iteration of the **for** loop will replace a ciphertext. We define the event $G_1(t, \mathsf{view}_t)$ in Algorithm 21 as "t is the smallest value among all t' that satisfy $\mathsf{ev}_{t,t'+1} = (\mathsf{ctxt}, x, c_x)$ with $x \in \mathsf{edges}_{d,i}$ and $\mathsf{maxcor}_{i,t'+2}(\mathsf{view}_t) < |x|$", and the event $G_2(t, \mathsf{view}_t)$ in Algorithm 22 as "t is the smallest value among all t' that satisfy $\mathsf{ev}_{t,t'+1} = (\mathsf{ctxt}, x, c_x)$ with $x \in \mathsf{edges}_{d,i}$ and $\mathsf{maxcor}_{i+1,t'+2}(\mathsf{view}_t) < |x|$".

Note that by Lemma 12, these definitions are in fact equivalent, i.e., $G_1(t, \mathsf{view}_t) = 1 \Leftrightarrow G_2(t, \mathsf{view}_t) = 1$. We will therefore in the following only speak of $G = G_1 = G_2$.

As mentioned above, we define F to be the event that the **repeat** loop in $R1_i$ or $R2_i$, respectively, is repeated more than B times, where B is defined in Claim 3.

As both definitions of G refer to the smallest t, it is obvious that the event G can occur at most once in either a run initiated using $I1$ and subsequent calls to $R1_i$, or a run initiated using $I2$ with subsequent calls to $R2_i$. Thus, G fulfills Items 1 and 2 from Lemma 10.

Let $P1_i$ be defined through $I1$ and $R1_i$ as in Lemma 10. Similarly, let $P2_i$ be defined through $I2$ and $R2_i$ as in Lemma 10. Let $P3_i$ be defined through $I1$, then sampling using $R1_i$ until the first occurrence of G, and then sampling using $R2_i$.

We note that $P1_i$ corresponds to sampling from $\widetilde{\mathcal{D}^{\mathsf{view}}_{0,b_{\mathsf{lkh}}}}$, $P2_i$ corresponds to sampling from $\widetilde{\mathcal{D}^{\mathsf{view}}_{i.1,b_{\mathsf{lkh}}}}$, and $P3_i$ corresponds to sampling from $\widetilde{\mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}}$.

Claim 3. For $B \coloneqq 2\mathcal{T} \cdot \ln(2\mathcal{T}/\gamma) \cdot \left(2^{d+2} + (3d+1) \cdot \mathcal{Q}_{\mathsf{corrupt}}\right)^2 \cdot (d+1)/\gamma$, the probability of F in $P1_i$, i.e. any run of the **repeat** *loop in Algorithm 21 exceeding B, is at most γ.*

Proof of Claim 3. By Lemma 6, with $g_t(x) = (\mathsf{maxcor}_{i,t}(x), \mathsf{len}(x), \mathsf{lastkey}_i(\mathsf{view}))$ and $h_t(\mathsf{view}) = \mathsf{pfx}_{t-1}(\mathsf{view})$ it holds that for any $\gamma \in (0,1]$ (thus in particular the γ from the theorem statement)

$$\Pr[\,\forall t \in [\mathcal{T}] : T_t^{\mathsf{rep}} \leq 2\mathcal{T} \cdot \ln(2\mathcal{T}/\gamma) \cdot |\mathcal{Y}|/\gamma\,] \;\geq\; 1-\gamma, \tag{15}$$

where T_t^{rep} denotes the number of runs of the repeat loop in the t-th iteration of the for loop, and $\mathcal{Y}$ denotes a set large enough to accommodate the range of any g_t. We note

$$\mathsf{len}(\mathsf{view}) \leq \mathcal{T}_{\max} \coloneqq \underbrace{2^{d+2}}_{\text{Initial Tree}} + \underbrace{\mathcal{Q}_{\mathsf{corrupt}} + 1}_{\text{Query Events}} + \underbrace{d \cdot \mathcal{Q}_{\mathsf{corrupt}}}_{\text{New Key Events}} + \underbrace{2 \cdot \mathcal{Q}_{\mathsf{corrupt}} \cdot (d-1)}_{\text{New ct Events}}$$

for any view resulting from a run of an adversary that makes at most $\mathcal{Q}_{\mathsf{corrupt}}$ corruption queries. This means that $\mathsf{len}(\mathsf{view})$ can take values up to $\mathcal{T}_{\max} = 2^{d+2} + (3d+1) \cdot \mathcal{Q}_{\mathsf{corrupt}}$. Furthermore, $\mathsf{maxcor}_{i,t}(\mathsf{view})$ takes values from 0 to d and $\mathsf{lastkey}_i(\mathsf{view})$ takes values from 0 to $\mathsf{len}(\mathsf{view})$.

Thus, $|\mathcal{Y}| \leq \left(2^{d+2} + (3d+1) \cdot \mathcal{Q}_{\mathsf{corrupt}}\right)^2 \cdot (d+1)$.

Plugging this into (15) yields

$$\Pr\left[\forall t \in [\mathcal{T}] : T_t^{\mathsf{rep}} \leq 2\mathcal{T} \cdot \ln(2\mathcal{T}/\gamma) \cdot \mathcal{T}_{\max}^2 \cdot (d+1)/\gamma\,\right] \geq\; 1-\gamma. \tag{16}$$

Thus, we can set

$$B \coloneqq 2\mathcal{T} \cdot \ln(2\mathcal{T}/\gamma) \cdot \mathcal{T}_{\max}^2 \cdot (d+1)/\gamma$$

where as before $\mathcal{T}_{\max} = 2^{d+2} + (3d+1) \cdot \mathcal{Q}_{\mathsf{corrupt}}$. □

We further note that by Lemma 5 instantiated with

$$h_t(\mathsf{view}_s) = \mathsf{pfx}_{t-1}(\mathsf{view}_s)$$
$$g_t(\mathsf{view}_s) = (\mathsf{maxcor}_{i,t+1}(\mathsf{view}_s), \mathsf{len}(\mathsf{view}_s), \mathsf{lastkey}_i(\mathsf{view}_s)),$$

the X_t defined by $P1_i$ are identically distributed to $\mathcal{D}^{\mathsf{view}}_{\mathsf{lkh},b_{\mathsf{lkh}}}$. Also by Lemma 5 instantiated with

$$h_t(\mathsf{view}_s) = \mathsf{pfx}_{t-1}(\mathsf{view}_s)$$
$$g_t(\mathsf{view}_s) = (\mathsf{maxcor}_{i+1,t+1}(\mathsf{view}_s), \mathsf{len}(\mathsf{view}_s), \mathsf{lastkey}_{i+1}(\mathsf{view}_s)),$$

X_t defined by $P1_{i+1}$ are identically distributed to $\mathcal{D}^{\mathsf{view}}_{\mathsf{lkh},b_{\mathsf{lkh}}}$.

Since before the occurrence of G in $P2_i$, the sampling of the X_t is equivalent to that of $P1_{i+1}$, the X_t up to G are distributed identically to the X_t in $P1_{i+1}$. It follows that G also fulfills the criteria Items 3 and 4 from Lemma 10.

Let B be chosen according to Claim 3. Then, the probability of F in $P1_i$ is at most γ. We now consider the probability of F in $P2_i$.

Claim 4. The probability of F in $P2_i$ is the same as the probability of $\perp \leftarrow \mathcal{D}^{\mathsf{view}}_{i,b_{\mathsf{lkh}}}$.

Proof of Claim 4. To see this, recall that by Lemma 12, the conditions for the **if** in Line 7 in Algorithm 19 for i are equivalent to the condition for the **if** in Line 3 in Algorithm 22. By the same argument, the end condition for the **repeat** loop is equivalent: The condition on $\mathsf{len}(\mathsf{view}_t)$ is independent of i and thus the same. By Lemma 13, if G occurred for X_t, $\mathsf{lastkey}_i(\mathsf{view}_{t'})$ and $\mathsf{lastkey}_{i+1}(\mathsf{view}_{t'})$ will be smaller than $t+1$ for any $t' > t$. Due to Lemma 11, a similar property holds for maxcor. □

Putting this together and using Lemma 10, the claim follows. □

Proposition 9. *$\mathcal{D}^{\mathsf{view}}_{T_d,b_{\mathsf{lkh}}}$ is independent of b_{lkh} and in particular $\mathrm{Adv}^{\mathsf{LKH},T_d}_{\mathsf{SKE},\mathcal{A},d}(\lambda) = 0$ for every LKH adversary $\mathcal{A}$.*

Proof of Proposition 9. Recall that b_{lkh} is only used when responding to the challenge query, which by assumption is the last query the adversary makes. Hence, neither the abort probability nor any of the events in $\mathsf{view}_{\mathcal{T}}$ before the very last events $(\mathsf{challenge}, k)$ and $(\mathsf{guess}, b_{\mathcal{A}})$ depend on b_{lkh}. The latter implies that the values of maxcor and $\mathsf{lastkey}$ (which are computed from corruption queries and ciphertext renewal events) as well as len are independent of b_{lkh} for all t. As the resampling procedure cuts off the tail of the view (including the last two events that may contain information about b_{lkh}) when resampling the views, no information about b_{lkh} is carried from view_{t-1} to view_t. Furthermore, since $\mathsf{edges}_{T_d} = \{0,1\}$, the final view $\mathsf{view}_{\mathcal{T}}$ does not contain any encryptions of the root key, and the root key k_ε is therefore independent of $\mathsf{view}_{\mathcal{T}}$. Thus, $\mathcal{A}$ has no advantage in distinguishing k_ε from a random independent key sampled by $\mathbf{Gen}(1^\lambda)$. □

We are now ready to prove the theorem.

Let $\mathcal{A}$ be an arbitrary LKH adversary running in time $t_{\mathcal{A}}$. First, $\mathcal{C}$ samples $i^* \leftarrow [T_d]$. It will then simulate the game $(i^* - 1).1$ or i^* depending on the challenge it gets from its own challenger. To generate a sample $\mathsf{view}_{\mathcal{T}}$, our $\mathsf{IND\text{-}CPA}$ adversary $\mathcal{C}$ modifies the procedure of Algorithm 19 by embedding an $\mathsf{IND\text{-}CPA}$ challenge in the if clause in the repeat loop, see Algorithm 23.

We briefly describe the algorithm. Let $x^* = \mathsf{edges}_{d,i-1} \Delta \mathsf{edges}_{d,i}$ be the edge that needs to either be pebbled or unpebbled in this game hop. The variable β indicates whether the former or the latter is the case. The core idea of Algorithm 23 is that it runs Algorithm 19, except that as the edge set it considers $\mathsf{edges}_{d,i} \cup \{x^*\}$, and when the edge x^* would be re-sampled during a rewinding to a ctxt event, the ciphertext is replaced with an IND-CPA challenge, where the permutation of the challenge messages is chosen depending on whether the edge is to be pebbled or unpebbled in the game hop (i.e. depending on β), implicitly setting the corresponding encryption key at the lower end of the edge to the challenge key. This is possible as for an edge to be pebbled or unpebbled, both other edges incident to its lower vertex need to be pebbled, i.e. both of the ciphertexts sitting on those edges need to have been replaced by encryptions of $\perp$ already, thus revealing nothing about the challenge key. Furthermore, any "honest" encryptions that need to be made with regard to this challenge key can be obtained by calling the LoR oracle provided by the $\mathsf{IND\text{-}CPA}$ challenger with two identical messages.

For any $x \in \mathsf{edges}_{d,i} \setminus \{x^*\}$, the algorithm $\mathcal{C}$ resamples a ciphertext of $\perp$ according to the same criteria as Algorithm 19. $\mathcal{C}$ outputs 0 if $\mathcal{A}$ succeeds and 1 else.

We have that for $b_{\mathsf{indcpa}} = 0$ and $i^* = i$ the modified algorithm samples from exactly the same distribution as Algorithm 19 on input $i - 1$ in the gray mode (and same $b_{\mathsf{lkh}} \in \{0,1\}, B \in \mathbb{N}$), and for $b_{\mathsf{indcpa}} = 1$ and $i^* = i$ from the same distribution as Algorithm 19 in the plain mode on input i (and same $b_{\mathsf{lkh}} \in \{0,1\}, B \in \mathbb{N}$). We obtain for the advantage of $\mathcal{C}$:

$$\begin{aligned}
\mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE},\mathcal{C}}(\lambda) &= \Pr[b_{\mathcal{C}} = b_{\mathsf{indcpa}}] - \frac{1}{2} \\
&= \frac{1}{2} \cdot (\Pr[b_{\mathcal{C}} = 0 \mid b_{\mathsf{indcpa}} = 0] + \Pr[b_{\mathcal{C}} = 1 \mid b_{\mathsf{indcpa}} = 1] - 1) \\
&= \frac{1}{2T_d} \cdot \sum_{i \in [T_d]} \left(\Pr\left[b_{\mathcal{C}} = 0 \middle| \begin{matrix} b_{\mathsf{indcpa}} = 0 \\ \wedge\, i^* = i \end{matrix} \right] - \Pr\left[b_{\mathcal{C}} = 0 \middle| \begin{matrix} b_{\mathsf{indcpa}} = 1 \\ \wedge\, i^* = i \end{matrix} \right] \right) \\
&= \frac{1}{2T_d} \cdot \sum_{i \in [T_d]} \left(\Pr[\mathbf{out}_{(i-1).1} = 1] - \Pr[\mathbf{out}_i = 1] \right) \\
&= \frac{1}{2T_d} \cdot \sum_{i \in [T_d]} \left(\mathrm{Adv}^{\mathsf{LKH},\,(i-1).1}_{\mathsf{SKE},\mathcal{A},d}(\lambda) - \mathrm{Adv}^{\mathsf{LKH},\,i}_{\mathsf{SKE},\mathcal{A},d}(\lambda) \right).
\end{aligned}$$

Algorithm 23: Variant of Algorithm 19 for sampling from $\mathcal{D}^{\mathsf{view}}_{i^*-1+b_{\mathsf{indcpa}},b_{\mathsf{lkh}}}$ given oracle access to an IND-CPA challenger with challenge bit b_{indcpa}. The functions $\mathsf{len}, \mathsf{maxcor}_{i^*,t}, \mathsf{lastkey}_{i^*}$ are described in the proof, B is as in Claim 3.

```
1  $i^* \leftarrow \{1,\ldots,T_d\}$          // guess which games need to be distinguished
2  $b_{\mathsf{lkh}} \leftarrow \{0,1\}$
3  Initialize $\iota := 0$                    // Counter for IND-CPA users
4  $y^*\|b^* := x^* := \mathsf{edges}_{d,i^*-1} \Delta \mathsf{edges}_{d,i^*}$, where $b^* \in \{0,1\}$    // differing edge
     index $x^* = y^*\|b^*$
5  $\beta := [x^* \in \mathsf{edges}_{d,i^*-1}]$     // bit $\beta$ indicates whether a pebble is added
     or removed in $i^*$th step
6  $\mathsf{view}_0 \leftarrow \mathcal{D}^{\mathsf{view}}_{\mathsf{lkh},b_{\mathsf{lkh}}}$
7  $\mathcal{T} := \mathsf{len}(\mathsf{view}_0)$              // Length of $\mathsf{view}_{\mathsf{lkh}}$ (in entries)
8  $t^* := \mathsf{lastkey}_{i^*}(\mathsf{view}_0)$      // Time when last key for $x^*$ is generated
9  for $t := 1$ to $\mathcal{T}$ do
10     Write $\mathsf{view}_{t-1} = (\mathsf{ev}_{t-1,1},\ldots,\mathsf{ev}_{t-1,\mathcal{T}})$
11     repeat              // Output $\perp$ if $B$ repetitions fail for this $t$
12         Rewind adversary to point $t$
13         if $t = t^*$ then
14             $\iota := \iota + 1$                  // Update current user index
15             $\mathsf{NU}()$              // Embed fresh IND-CPA challenge key
16         end
17         if $\mathsf{ev}_{t-1,t} = (\mathsf{ctxt}, x, c_x)$ with $x \in \mathsf{edges}_{d,i^*} \cup \{x^*\}$ and
             $\mathsf{maxcor}_{i^*,t+1}(\mathsf{view}_{t-1}) < |x|$ then
                   // Checks if $\mathsf{ev}_{t,t}$ defines a ciphertext to be pebbled
18             if $x = x^*$ then
19                 Set $k^*_\beta := k_{y^*}$, $k^*_{1-\beta} := \perp$
20                 $c^* \leftarrow \mathsf{LoR}(\iota, k_0, k_1)$ from IND-CPA challenger          // Fresh
                     IND-CPA challenge ciphertext
21                 Set $\mathsf{ev}_{t,t} := (\mathsf{ctxt}, x^*, c^*)$   // Replace $c_x$ with $c^*$ in $\mathsf{view}_{t-1}$
22             else
23                 Sample fresh $c^\perp_x \leftarrow \mathbf{Enc}(k_x, \perp)$   // Fresh dummy ciphertext
24                 Set $\mathsf{ev}_{t,t} := (\mathsf{ctxt}, x, c^\perp_x)$   // Replace $c_x$ with $c^\perp_x$ in $\mathsf{view}_{t-1}$
25             end
26             Resample from point $t+1$ to obtain
                 $\mathsf{view}_t = (\mathsf{ev}_{t-1,0},\ldots,\mathsf{ev}_{t-1,t-1},\mathsf{ev}_{t,t},\mathsf{ev}_{t,t+1},\ldots,\mathsf{ev}_{t,\tau})$
27         else
28             Resample from point $t$ to obtain
                 $\mathsf{view}_t = (\mathsf{ev}_{t-1,0},\ldots,\mathsf{ev}_{t-1,t-1},\mathsf{ev}_{t,t},\ldots,\mathsf{ev}_{t,\tau})$ but with the
                 following change: for all $t' > t^*$ with $\mathsf{ev}_{t,t'} = (\mathsf{ctxt}, x^*, c_{x^*})$
                 generate $c_{x^*} \leftarrow \mathsf{LoR}(\iota, k_{y^*}, k_{y^*})$
29         end
30     until $\mathsf{maxcor}_{i^*,t+1}(\mathsf{view}_t) = \mathsf{maxcor}_{i^*,t+1}(\mathsf{view}_{t-1})$ and
         $\mathsf{len}(\mathsf{view}_t) = \mathsf{len}(\mathsf{view}_{t-1})$ and $\mathsf{lastkey}_{i^*}(\mathsf{view}_t) = \mathsf{lastkey}_{i^*}(\mathsf{view}_{t-1})$
31 end
32 return $[\mathbf{out}_{\mathcal{A}}(\mathsf{view}_{\mathcal{T}}) = b_{\mathsf{lkh}}]$
```

By Proposition 8 we have for all $i \in [T_d]$

$$\mathrm{Adv}^{\mathsf{LKH},\,(i-1).1}_{\mathsf{SKE},\mathcal{A},d}(\lambda) - \mathrm{Adv}^{\mathsf{LKH},\,i-1}_{\mathsf{SKE},\mathcal{A},d}(\lambda) \geq -\gamma$$

and hence we obtain

$$\begin{aligned}\mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE},\mathcal{C}}(\lambda) &= \frac{1}{2} \cdot \frac{1}{T_d} \cdot \sum_{i \in [T_d]} \left(\mathrm{Adv}^{\mathsf{LKH},\,(i-1).1}_{\mathsf{SKE},\mathcal{A},d}(\lambda) - \mathrm{Adv}^{\mathsf{LKH},\,i}_{\mathsf{SKE},\mathcal{A},d}(\lambda)\right) \\ &\geq \frac{1}{2} \cdot \frac{1}{T_d} \cdot \sum_{i \in [T_d]} \left(\mathrm{Adv}^{\mathsf{LKH},\,i-1}_{\mathsf{SKE},\mathcal{A},d}(\lambda) - \mathrm{Adv}^{\mathsf{LKH},\,i}_{\mathsf{SKE},\mathcal{A},d}(\lambda) - \gamma\right) \\ &\geq \frac{1}{2} \cdot \frac{1}{T_d} \cdot \left(\mathrm{Adv}^{\mathsf{LKH}}_{\mathsf{SKE},\mathcal{A},d}(\lambda) - \mathrm{Adv}^{\mathsf{LKH},\,T_d}_{\mathsf{SKE},\mathcal{A},d}(\lambda)\right) - \frac{\gamma}{2}.\end{aligned}$$

Plugging in Proposition 9 yields

$$\mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE},\mathcal{C}}(\lambda) \geq \frac{1}{2} \cdot \frac{1}{T_d} \cdot \mathrm{Adv}^{\mathsf{LKH}}_{\mathsf{SKE},\mathcal{A},d}(\lambda) - \frac{\gamma}{2}.$$

For the runtime analysis we see that the **for** loop is called $\mathcal{T}$ times and the **repeat** loop is called at most B times. During each run of the **repeat** loop, the adversary $\mathcal{A}$ is called once. This yields the runtime given in the theorem statement. □

References

[Bad+16] Christoph Bader, Tibor Jager, Yong Li, and Sven Schäge. "On the Impossibility of Tight Cryptographic Reductions". In: *EUROCRYPT 2016, Part II*. 2016.

[BBR18] Karthikeyan Bhargavan, Richard Barnes, and Eric Rescorla. *TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups A protocol proposal for Messaging Layer Security (MLS)*. Research Report. Inria Paris, 2018. URL: https://hal.inria.fr/hal-02425247.

[BDT22] Alexander Bienstock, Yevgeniy Dodis, and Yi Tang. "Multicast Key Agreement, Revisited". In: *Topics in Cryptology – CT-RSA 2022*. 2022.

[Bel+97] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. "A Concrete Security Treatment of Symmetric Encryption". In: *38th FOCS*. 1997.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. "Functional Signatures and Pseudorandom Functions". In: *PKC 2014*. 2014.

[BK10] Zvika Brakerski and Yael Tauman Kalai. *A Framework for Efficient Signatures, Ring Signatures and Identity Based Encryption in the Standard Model*. Cryptology ePrint Archive, Report 2010/086. 2010.

[BW13] Dan Boneh and Brent Waters. "Constrained Pseudorandom Functions and Their Applications". In: *ASIACRYPT 2013, Part II*. 2013.

[Can+01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. "Black-box concurrent zero-knowledge requires Omega (log n) rounds". In: *33rd ACM STOC*. 2001.

[Can+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*. 1999.

[Cor02] Jean-Sébastien Coron. "Optimal Security Proofs for PSS and Other Signature Schemes". In: *EUROCRYPT 2002.* 2002.

[Dav+20] Alex Davidson, Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. "Adaptively Secure Constrained Pseudorandom Functions in the Standard Model". In: *CRYPTO 2020, Part I.* 2020.

[FF13] Marc Fischlin and Nils Fleischhacker. "Limitations of the Meta-reduction Technique: The Case of Schnorr Signatures". In: *EUROCRYPT 2013.* 2013.

[FS87] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *CRYPTO'86.* 1987.

[Fuc+14] Georg Fuchsbauer, Momchil Konstantinov, Krzysztof Pietrzak, and Vanishree Rao. "Adaptive Security of Constrained PRFs". In: *ASIACRYPT 2014, Part II.* 2014.

[GGM84a] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. "How to Construct Random Functions (Extended Abstract)". In: *25th FOCS.* 1984.

[GGM84b] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. "On the Cryptographic Applications of Random Functions". In: *CRYPTO'84.* 1984.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. "A Digital Signature Scheme Secure Against Adaptive Chosen-message Attacks". In: *SIAM Journal on Computing* 2 (1988).

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Prove all NP-Statements in Zero-Knowledge, and a Methodology of Cryptographic Protocol Design". In: *CRYPTO'86.* 1987.

[HJK12] Dennis Hofheinz, Tibor Jager, and Edward Knapp. "Waters Signatures with Optimal Security Reduction". In: *PKC 2012.* 2012.

[HK08] Dennis Hofheinz and Eike Kiltz. "Programmable Hash Functions and Their Applications". In: *CRYPTO 2008.* 2008.

[HW09] Susan Hohenberger and Brent Waters. "Short and Stateless Signatures from the RSA Assumption". In: *CRYPTO 2009.* 2009.

[Jaf+17] Zahra Jafargholi, Chethan Kamath, Karen Klein, Ilan Komargodski, Krzysztof Pietrzak, and Daniel Wichs. "Be Adaptive, Avoid Overcommitting". In: *CRYPTO 2017, Part I.* 2017.

[Kam+21] Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. "The Cost of Adaptivity in Security Games on Graphs". In: *TCC 2021, Part II.* 2021.

[Kia+13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. "Delegatable pseudorandom functions and applications". In: *ACM CCS 2013.* 2013.

[Kle+21] Karen Klein, Guillermo Pascual-Perez, Michael Walter, Chethan Kamath, Margarita Capretto, Miguel Cueto, Ilia Markov, Michelle Yeo, Joël Alwen, and Krzysztof Pietrzak. "Keep the Dirt: Tainted TreeKEM, Adaptively and Actively Secure Continuous Group Key Agreement". In: *2021 IEEE Symposium on Security and Privacy.* 2021.

[KLX22] Julia Kastner, Julian Loss, and Jiayu Xu. "The Abe-Okamoto Partially Blind Signature Scheme Revisited". In: *Advances in Cryptology – ASIACRYPT 2022.* 2022.

[Kuc+20] Veronika Kuchta, Amin Sakzad, Damien Stehlé, Ron Steinfeld, and Shifeng Sun. "Measure-Rewind-Measure: Tighter Quantum Random Oracle Model Proofs for One-Way to Hiding and CCA Security". In: *EUROCRYPT 2020, Part III.* 2020.

[LW14] Allison B. Lewko and Brent Waters. “Why Proving HIBE Systems Secure Is Difficult”. In: *EUROCRYPT 2014*. 2014.

[NY89] Moni Naor and Moti Yung. “Universal One-Way Hash Functions and their Cryptographic Applications”. In: *21st ACM STOC*. 1989.

[Pan07] Saurabh Panjwani. “Tackling Adaptive Corruptions in Multicast Encryption Protocols”. In: *TCC 2007*. 2007.

[PS96] David Pointcheval and Jacques Stern. “Security Proofs for Signature Schemes”. In: *EUROCRYPT'96*. 1996.

[RK99] Ransom Richardson and Joe Kilian. “On the Concurrent Composition of Zero-Knowledge Proofs”. In: *EUROCRYPT'99*. 1999.

[Wat05] Brent R. Waters. “Efficient Identity-Based Encryption Without Random Oracles”. In: *EUROCRYPT 2005*. 2005.

[WGL00] Chung Kei Wong, Mohamed G. Gouda, and Simon S. Lam. “Secure group communications using key graphs”. In: *IEEE/ACM Trans. Netw.* 1 (2000). URL: `https://doi.org/10.1109/90.836475`.

[WHA98] D. M. Wallner, E. J. Harder, and R. C. Agee. *Key Management for Multicast: Issues and Architectures*. Internet Draft. 1998.