Salvaging Indifferentiability in a Multi-stage Setting

Arno Mittelbach

Darmstadt University of Technology, Germany
www.cryptoplexity.de
arno.mittelbach@cased.de

Abstract. Ristenpart, Shacham and Shrimpton (Eurocrypt 2011) recently presented schemes which are provably secure in the random-oracle model (ROM), but easily broken if the random oracle is replaced by typical indifferentiable hash constructions such as chop-MD or prefix-free-MD. They found that the indifferentiability framework, due to Maurer, Renner and Holenstein (TCC 2004), does not necessarily allow composition in multi-stage settings, that is, settings consisting of multiple disjoint adversarial stages. On the positive side, they prove that the non-adaptive chosen distribution attack (CDA) game of Bellare et al. (Asiacrypt 2009), a multi-stage game capturing the security of deterministic encryption schemes, remains secure if the random oracle is implemented by an NMAC-like hash function.

In this paper we introduce a framework to work with the indifferentiability notion in multi-stage scenarios. For this we provide a model for iterative hash functions which is general enough to cover not only NMAC-like functions, but also functions such as chop-MD or even hash trees. We go on to define a property on multi-stage games called *unsplittability* which intuitively captures that adversaries cannot split the computation of a single hash value over several stages. We present a composition theorem for unsplittable multi-stage games which generalizes the single-stage composition theorem for indifferentiable hash functions. We then show that the CDA game (adaptive or non-adaptive) is unsplittable for *any* iterative hash function (thereby extending the preliminary results by Ristenpart et al.). Finally, we prove that the *proof-of-storage* game presented by Ristenpart et al. as a counterexample to the general applicability of the indifferentiability framework is unsplittable for any multi-round iterative hash function, such as Liskov's Zipper Hash (SAC 2006).

Keywords. Hash functions, Random oracle, Indifferentiability, Multi-stage

Contents

$\underline{\mathbf{CDA}^{H^{\mathtt{h}},\mathcal{A}_1,\mathcal{A}_2}_{\mathcal{AE}}(1^\lambda)}$

$b \leftarrow \{0,1\}$
$(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$
$(\mathbf{m_0}, \mathbf{m_1}, \mathbf{r}) \leftarrow \mathcal{A}_1^{\mathtt{h}}(1^\lambda)$
$\mathbf{c} \leftarrow \mathcal{E}^{H^{\mathtt{h}}}(pk, \mathbf{m_b}; \mathbf{r})$
$b' \leftarrow \mathcal{A}_2^{\mathtt{h}}(pk, \mathbf{c})$
return $(b = b')$

$\underline{\mathbf{CRP}^{H^{\mathtt{h}},\mathcal{A}_1,\mathcal{A}_2}_{p,s}(1^\lambda)}$

$M \leftarrow \{0,1\}^p$
$st \leftarrow \mathcal{A}_1^{\mathtt{h}}(M, 1^\lambda)$
If $|st| > n$ then return false;
$C \leftarrow \{0,1\}^c$
$Z \leftarrow \mathcal{A}_2^{\mathtt{h}}(st, C)$
return $(Z = H^{\mathtt{h}}(M||C))$

Figure 1: Security Games. The chosen distribution attack (CDA) game [6] capturing security in deterministic encryption schemes [5] and the proof-of-storage challenge-response game (CRP) due to Ristenpart et al. [31].

1 Introduction

The notion of indifferentiability, introduced by Maurer, Renner and Holenstein (MRH) [26] can be regarded as a generalization of indistinguishability tailored for situations where internal state is publicly available. It has found wide applicability in the domain of hash functions which are usually built from a fixed-length compression function together with a scheme that describes how arbitrarily long messages are to be processed [27, 18, 32, 23, 11]. An honest user would always use such a hash function as specified. An adversary, on the other hand, might use its knowledge on the internal state (i.e., intermediate compression-function evaluations) if it gains any advantage. The MRH composition theorem [26] allows to reduce security in the random oracle model (ROM) [8] to the security of compression functions in settings where the random oracle is instantiated by a hash function which is indifferentiable from a random oracle. Several indifferentiable hash constructions have since been presented [16, 23] and also various finalists of NIST's SHA-3 competition [29]—including the winner Keccak [12]—enjoy proofs of indifferentiability [15, 2, 28, 13, 14].

Recently, Ristenpart, Shacham and Shrimpton gave the somewhat surprising result that the MRH composition theorem only holds in single-stage settings and does not necessarily extend to multi-stage settings where disjoint adversaries are split over several stages [31]. As counterexample they present a simple *challenge-response game* (CRP, depicted in Figure 1): a file server that is given a file M can be engaged in a simple proof-of-storage protocol. We say the adversary wins if it can successfully complete the protocol, while only storing a short state st (with $|st| \ll |M|$). For the proof-of-storage protocol, the client prepares a challenge C that it sends to the server which has to respond with hash value $\mathcal{H}(M||C)$. The protocol can easily be proven secure in the ROM since, without access to file M, it is highly improbable for the server to correctly guess the hash value $\mathcal{H}(M||C)$. The server can, however, "cheat" if the random oracle is replaced by the indifferentiable chop-Merkle-Damgård construction (chop-MD) presented in [16]. Chop-MD is very similar to the original Merkle-Damgård construction[1] [27, 18] with but only outputs the first $n - k$ bits of the final compression function evaluation as hash value. In this case, a corrupt server can win the CRP game by utilizing the compression function underlying the hash construction. On receiving file M, the server (adversary $\mathcal{A}_1$ in Figure 1) computes hash value $H^{\mathtt{h}}(M)$ manually (we write $H^{\mathtt{h}}$ instead of $\mathcal{H}$ to denote that the hash function is built from an underlying function $\mathtt{h}$), solely using the underlying compression function $\mathtt{h}$, without chopping off any bits from the last compression-function output. It throws away file M and only stores the output of the last compression-function call (i.e., passes on this intermediate value as its state st). On receiving a challenge C, the server (adversary $\mathcal{A}_2$) can then return the first $n - k$ bits of value $\mathtt{h}(C, st)$ where $\mathtt{h}$ denotes the underlying compression function. If we ignore padding (assuming a padding function only makes notation more cumbersome, but does not prevent the attack) it holds that

$$\mathtt{h}(C, st)_{1,\ldots,n-k} = H^{\mathtt{h}}(M||C)$$

and the adversary wins with probability 1.

To circumvent the problem of composition in multi-stage settings, Ristenpart et al. [31] propose a stronger form of indifferentiability called *reset indifferentiability*, which intuitively states that simulators

[1] The Merkle-Damgård hash function $H^{\mathtt{h}}(m_1, \ldots, m_\ell)$ is computed as $H^{\mathtt{h}}(m_1, \ldots, m_\ell) := \mathtt{h}(m_\ell, x_{\ell-1})$ where $x_0 := \mathcal{IV}$ is some initialization vector and $x_i := \mathtt{h}(m_i, x_{i-1})$ is computed as the compression function evaluated on the current message block and the last chaining value.

(which are used to simulate a compression function for a random oracle) must be stateless. While this notion would allow composition in any setting, no domain extender can fulfill this stronger form of indifferentiability [19, 25]. Demay et al. [19] present a second variant of indifferentiability called *resource-restricted indifferentiability* which models simulators with explicit memory restrictions and which lies somewhere in between plain indifferentiability and reset indifferentiability. However, they do not present any positive results such as constructions that achieve any form of resource-restricted indifferentiability or security games for which a resource-restricted construction allows composition.

Ristenpart et al. also present a positive result [31] in this direction. They examine the (multi-stage) non-adaptive chosen-distribution attack (CDA) game [6], depicted in Figure 1, which captures a security notion for deterministic public-key encryption schemes [5], where the randomness does not have sufficient min-entropy. In the CDA game, the first-stage adversary $\mathcal{A}_1$ outputs two message vectors $\mathbf{m_0}$ and $\mathbf{m_1}$ together with a randomness vector $\mathbf{r}$ which, together, must have sufficient min-entropy independent of the hash functionality. According to a secret bit b one of the two message vectors is encrypted and given, together with the public key, to the second-stage adversary $\mathcal{A}_2$. The adversary wins if it correctly guesses b.

For the non-adaptive CDA game, Ristenpart et al. give a direct security proof for the subclass of indifferentiable hash functions of the NMAC-type[7], i.e., hash functions of the form $H^{\mathtt{h}}(M) := \mathtt{g}(f^{\mathtt{h}}(M))$ where functions $\mathtt{g}$ and $\mathtt{h}$ are independent fixed-length random oracles. Note that, while this covers some hash functions of interest, it does not, for example, cover chop-MD functions (like SHA-2 for certain parameter settings) or Keccak (aka. SHA-3).

Contributions. In this paper we build on this last idea and ask for a more general way of working with indifferentiability in a multi-stage setting. Our first contribution is a model of iterative hash functions that is general enough to cover many practical functions: for example NMAC-like functions, Merkle-Damgård variants such as chop-MD, prefix-free MD or even the plain Merkle-Damgård scheme but also more elaborate schemes such as hash trees (Section 3).

Using our model of iterated hash functions we characterize a property of multi-stage games called *unsplittability* (Section 4). If a game is *unsplittable* for an iterative hash function $H^{\mathtt{h}}$, this intuitively says that an adversary does not gain any advantage in splitting up the computation of a hash value over several distinct stages (as opposed to in the CRP game from the introduction) during the game when function $H^{\mathtt{h}}$ is used. Formally, we call a game *unsplittable* for an iterative hash function $H^{\mathtt{h}}$ if any adversary can be transformed in such a way that certain *bad* queries to the underlying function $\mathtt{h}$ only occur with small probability while the game's outcome only changes negligibly. In Section 5 we then give a composition theorem for games that are *unsplittable*. Analogously to the MRH-theorem, our composition theorem intuitively says that if a game is *unsplittable* for an indifferentiable hash function $H^{\mathtt{h}}$, then any adversary against the game with $H^{\mathtt{h}}$ can be transformed into an adversary against the game with a random oracle $\mathcal{R}$. We stress that the hash function just has to be plain indifferentiable. In other words, security proofs in the random-oracle model carry over if the random oracle is replaced by this particular hash construction $H^{\mathtt{h}}$.

Furthermore, we show that the CDA game (non-adaptive *and* adaptive) is *unsplittable* for any iterative hash function, thereby strengthening the preliminary results by Ristenpart et al. [31] (Section 4.1). For the adaptive version of the CDA game we introduce a notion of key extractability for public-key encryption schemes (called PK-EXT), that we believe to be of independent interest (Appendix F). PK-EXT can be regarded as an adaption of the key indistinguishability notion by Bellare et al. [4] for the setting of deterministic encryption. Finally, we show that the CRP game discussed in the introduction is *unsplittable* for any multi-round iterative hash-function (Section 4.1).

2 Preliminaries

2.1 Notation

If $n \in \mathbb{N}$ is a natural number then by 1^n we denote the unary representation and by $\langle n \rangle_\ell$ the binary representation of n (using ℓ bits). By $[n]$ we denote the set $\{1, 2, \ldots, n\}$. By $\{0,1\}^n$ we denote the set of all bit strings of length n while $\{0,1\}^*$ denotes the set of all finite bit strings. For bit strings $m, m' \in \{0,1\}^*$ we denote by $m||m'$ their concatenation. If $\mathcal{M}$ is a set then by $m \leftarrow \mathcal{M}$ we denote that

m was sampled uniformly from $\mathcal{M}$. If $\mathcal{A}$ is an algorithm then by $X \leftarrow \mathcal{A}(m)$ we denote that X was output by algorithm $\mathcal{A}$ on input m. As usual $|\mathcal{M}|$ denotes the cardinality of set $\mathcal{M}$ and $|m|$ the length of bit string m. Logarithms are to base 2. By $\mathrm{H}_\infty(X)$ we denote the min-entropy of variable X, defined as

$$\mathrm{H}_\infty(X) := \min_{x \in \mathrm{Supp}(X)} \log(1/\Pr[X = x]) \ .$$

We assume that any algorithm, game, etc. is implicitly given a security parameter as input, even if not explicitly stated. We call an algorithm *efficient* if its run-time is polynomial in the security parameter.

A hash function is formally defined as a keyed family of functions $\mathcal{H}(1^\lambda)$ where each key k defines a function $H_k : \{0,1\}^* \to \{0,1\}^n$. "Practical" hash functions are usually built via domain extension from a fixed-length function $\mathtt{h} : \{0,1\}^{d \times k} \to \{0,1\}^s$ that is iterated through an iteration scheme H to process arbitrarily long inputs [27, 18, 32, 23, 3, 21, 34, 11, 20], with widely varying specifications. The underlying function $\mathtt{h}$ is often build from block-ciphers (e.g., using the Davies–Meyer construction [33]) and is often compressing. As exception to this rule, the Sponge construction [12] (the design principle behind SHA-3, aka. Keccak [11]) iterates a permutation instead of a compression function. For the results in this paper this difference is mostly irrelevant: we model the underlying function $\mathtt{h}$ as an ideal function, that is a function taken uniformly at random from the space of all functions with the given domain and target space. Our results, thus, hold up-to a distinguishing probability of an ideal permutation from a random function, in case the iterated function were a permutation. We will simply speak of the *underlying* function when talking about $\mathtt{h}$ and write $\mathtt{h} : \{0,1\}^d \times \{0,1\}^k \to \{0,1\}^s$ to denote that $\mathtt{h}$ takes two distinct inputs, one usually corresponding to message blocks, the other corresponding to intermediate values.

2.2 Indifferentiability

A hash function is called indifferentiable from a random oracle if no efficient distinguisher can decide whether it is talking to the hash function and its ideal compression function (modeled as a fixed-length random oracle) or to an actual random oracle and a simulator. We now give the definition of indifferentiability from [16]:

Definition 2.1. *A hash construction $H^{\mathtt{h}} : \{0,1\}^* \to \{0,1\}^n$, with black-box access to an ideal function $\mathtt{h} : \{0,1\}^d \times \{0,1\}^k \to \{0,1\}^s$, is called $(t_\mathcal{D}, t_\mathcal{S}, q, \epsilon)$ indifferentiable from a random oracle $\mathcal{R}$ if there exists an efficient simulator $\mathcal{S}^\mathcal{R}$ such that for any distinguisher $\mathcal{D}$ it holds that*

$$\left| \Pr\left[\mathcal{D}^{H^{\mathtt{h}},\mathtt{h}}(1^\lambda) = 1 \right] - \Pr\left[\mathcal{D}^{\mathcal{R},\mathcal{S}^\mathcal{R}}(1^\lambda) = 1 \right] \right| \leq \epsilon$$

where the simulator runs in time at most $t_\mathcal{S}$, and the distinguisher runs in time at most $t_\mathcal{D}$ and makes at most q queries. We say $H^{\mathtt{h}}$ is (computationally) indifferentiable from $\mathcal{R}$ if ϵ is a negligible function in the security parameter λ and $t_\mathcal{D}$ and $t_\mathcal{S}$ are polynomials (in λ).

We define the advantage of a distinguisher $\mathcal{D}$ with respect to a simulator $\mathcal{S}$ in the indifferentiability game as

$$\mathbf{Adv}^{\mathrm{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}) = \left| \Pr\left[\mathcal{D}^{H^{\mathtt{h}},\mathtt{h}}(1^\lambda) = 1 \right] - \Pr\left[\mathcal{D}^{\mathcal{R},\mathcal{S}^\mathcal{R}}(1^\lambda) = 1 \right] \right| \ .$$

We sometimes speak of the *real world* when meaning that the distinguisher is connected to hash function $H^{\mathtt{h}}$ and underlying function $\mathtt{h}$ and of the *ideal world* when it is talking to random oracle $\mathcal{R}$ and simulator $\mathcal{S}^\mathcal{R}$.

Remark. This notion of indifferentiability is also sometimes called *strong* indifferentiability. The difference between strong and weak indifferentiability is the order of quantifiers. For weak indifferentiability (as originally presented in [26]) the simulator may depend on the distinguisher, which is also sufficient for composition. In this paper we only consider the strong setting, as to the best of our knowledge, all hash constructions that have a proof of indifferentiability have been proved in the strong setting.

Indifferentiability in Multi-Stage Settings. Intuitively the MRH composition theorem [26, 16] for indifferentiable hash functions says that a proof given in the ROM can be reduced to the security of the underlying function when the random oracle is implemented by an indifferentiable hash function. The idea is that the simulator can simulate ideal function $\mathtt{h}$ for an adversary relative to a random oracle. Thus, from an adversary $\mathcal{A}$ against a scheme in the real world, we can build a new adversary $\mathcal{B} := \mathcal{A}^{\mathcal{S}^{\mathcal{R}}}$ which is successful against attacking the scheme in the random oracle model.

In multi-stage settings (such as the proof-of-storage example from the introduction) the difference is that we deal with multiple disjoint adversaries $\mathcal{A}_1, \ldots, \mathcal{A}_m$ that do not share their entire state. When applying the above idea to this setting we would, hence, need to give each adversary access to *its own* instance of simulator $\mathcal{S}$. As these multi-stage adversaries do not share their entire state (and so do the corresponding simulators) the simulators may not be able to perfectly simulate the underlying function $\mathtt{h}$; this guarantee given by a proof of indifferentiability only holds if the simulator sees all the queries to the compression function instead of just a fraction of them.

Reset indifferentiability as introduced by Ristenpart et al. [31] extends Definition 2.1 by allowing the distinguisher to reset the simulator at arbitrary times, which intuitively means that the simulator must be stateless [30, 19]. While this allows composition in arbitrary settings (as the simulator is stateless each adversarial stage $\mathcal{A}_i$ can have their own instance of the simulator) no hash construction based on domain extension can fulfill this stronger notion [19, 25]; thus effectively ruling out every hash construction that we know of today.

2.3 Game Playing

For the upcoming discussion we use the game-playing technique [9, 31]. We give here a brief overview of the notation used and present a self-contained introduction in Appendix B.

A game $G^{\mathcal{F},\mathcal{A}_1,\ldots,\mathcal{A}_m}$ gets access to adversarial procedures $\mathcal{A}_1, \ldots, \mathcal{A}_m$ and to one or more so called functionalities $\mathcal{F}$ which are collections of two procedures $\mathcal{F}$.hon and $\mathcal{F}$.adv, with suggestive names "honest" and "adversarial". Adversaries (i.e., adversarial procedures) access a functionality $\mathcal{F}$ via the interface exported by $\mathcal{F}$.adv, while all other procedures access the functionality via $\mathcal{F}$.hon. In our case, functionalities are exclusively hash functions which will be instantiated with iterative hash constructions $H^{\mathtt{h}}$. The adversarial interface exports the underlying function $\mathtt{h}$, while the honest interface exports plain access to $H^{\mathtt{h}}$. We thus, instead of writing $\mathcal{F}$.hon and $\mathcal{F}$.adv directly refer to $H^{\mathtt{h}}$ and $\mathtt{h}$, respectively. Adversarial procedures can only be called by the game's **main** procedure.

By $G^{\mathcal{F},\mathcal{A}_1,\ldots,\mathcal{A}_m} \Rightarrow y$ we denote that the game outputs value y. Games are random variables over the entire random coins of the game and the adversarial procedures. For functionalities $\mathcal{F}$ and $\mathcal{F}'$ and adversaries $\mathcal{A}_1, \ldots, \mathcal{A}_m$ and $\mathcal{A}'_1, \ldots, \mathcal{A}'_m$, we can thus consider the distance between the two random variables. We say two games are ϵ-close if for all values y it holds that

$$\Pr\left[\, G^{\mathcal{F},\mathcal{A}_1,\ldots,\mathcal{A}_m} \Rightarrow y \,\right] \leq \Pr\left[\, G^{\mathcal{F}',\mathcal{A}'_1,\ldots,\mathcal{A}'_m} \Rightarrow y \,\right] + \epsilon$$

Functionality Respecting Games. Let $\mathcal{LG}$ be the set of all functionality-respecting games [31]. A game is called *functionality respecting* if only adversarial procedures can call the adversarial interface of functionalities. Note that this restriction is quite natural if a game is used to specify a security goal in the random oracle model since random oracles do not provide any adversarial interface.

3 A Model for Iterative Hash Functions

In the following we present a new model for iterated hash functions that is general enough to properly capture many different types of constructions (ranging from the plain Merkle-Damgård over variants such as chop-MD or Sponge to more complex constructions such as NMAC, HMAC [7] or even hash trees; also see Appendix A.2) while, at the same time, allowing to reason about indifferentiability [26] in multi-stage scenarios. Furthermore, we believe that our characterization may be useful in different settings as it allows to reason about a large class of hash functions simultaneously.

We model hash functions $H^{\mathtt{h}}$ as directed graphs, where each message M is mapped to a graph. A generic algorithm $\mathtt{EVAL}^{\mathtt{h}}$ with access to an oracle $\mathtt{h}$ and input the execution graph for M can then compute value $H^{\mathtt{h}}(M)$.

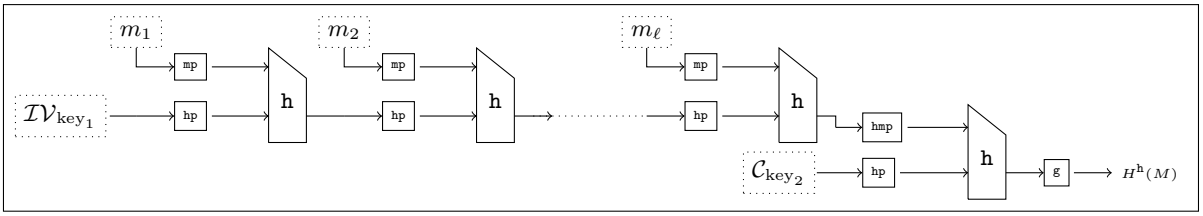

Figure 2: NMAC execution graph for message $m_1 \| \ldots \| m_\ell := M$. Value $\mathcal{IV}_{\text{key}_1}$ is an initialization vector representing the first key in the NMAC-construction. Value $\mathcal{C}_{\text{key}_2}$ is a constant representing the second key. The difference between initialization vectors and constants is that constants are used within the execution graph, i.e., in conjunction with interim values, while initialization vectors are used at the beginning of the graph.

Execution Graphs - An Introduction. Figure 2 presents the *execution graph* for a message $M := m_1 \| \ldots \| m_\ell$ for the NMAC construction [7] (further examples are given in Appendix A.2). The idea is to capture the evaluation of a hash function $H^{\mathtt{h}}$ as a directed graph. For each input message M the corresponding execution graph represents how the hash value would be computed relative to some oracle $\mathtt{h}$. (This execution graph can be constructed without access to the underlying ideal function $\mathtt{h}$.) Nodes in the graph are either *value-nodes* or *function-nodes*. A value node (indicated by dotted boxes) does not have ingoing edges and the outgoing edge is usually labeled with the node's label. Function nodes represent functions and the outgoing edges are labeled with the result of the evaluation of the corresponding function taking the labels of the ingoing edges as input. An $\mathtt{h}$-node represents the evaluation of the underlying ideal function $\mathtt{h}$. The outgoing edges can thus only be labeled relative to $\mathtt{h}$. Nodes labeled $\mathtt{mp}, \mathtt{hp}$ or $\mathtt{hmp}$ correspond to preprocessing functions (defined by the hash construction) which ensure that the input to the next $\mathtt{h}$-node is of correct length: $\mathtt{mp}$ processes message blocks, $\mathtt{hp}$ processes $\mathtt{h}$-outputs and $\mathtt{hmp}$, likewise, processes the output of $\mathtt{h}$-nodes but such that it can go into the "message slot" of an $\mathtt{h}$-node (see Figure 2). An execution graph contains exactly one $\mathtt{g}$-node with an unbound outgoing edge which corresponds to an (efficiently) computable transformation such as the identity or truncation. Assume that $\mathfrak{eg}$ is the execution graph for a message $M \in \{0,1\}^*$. Then we can formalize the computation of hash value $H^{\mathtt{h}}(M)$ with underlying function $\mathtt{h}$ by a deterministic algorithm $\mathtt{EVAL}^{\mathtt{h}}(\mathfrak{eg})$ which repeats the following steps: search for a node with no input edges or where all input edges are labeled. Compute the corresponding function (if it is an $\mathtt{h}$-node, call the provided $\mathtt{h}$-oracle), remove the node and label all outgoing edges with the resulting value. The label of the single unbound outgoing edge of the $\mathtt{g}$-node is the resulting hash value.

Formalizing Hash Functions as Directed Graphs. We now formalize the above concept. For this let $\mathtt{pad} : \{0,1\}^* \to (\{0,1\}^b)^+$ be padding function (e.g. Merkle-Damgård strengthening [18, 27]) that maps strings to multiples of block size b. Let $\mathtt{mp} : \{0,1\}^* \to \{0,1\}^d$ be "preprocessing" function used to adapt message blocks. Additionally, let $\mathtt{hp} : \{0,1\}^* \to \{0,1\}^k$ and $\mathtt{hmp} : \{0,1\}^* \to \{0,1\}^d$ be two "preprocessing" functions that allow to adapt intermediate hash values. We assume that $\mathtt{pad}, \mathtt{mp}, \mathtt{hp}$, and $\mathtt{hmp}$ are efficiently computable injective and also efficiently invertible. Note that for many schemes these functions will be the identity function and $b = d$ and $s = k$. Let $\mathtt{g} : \{0,1\}^s \to \{0,1\}^n$ be an efficiently computable transformation (such as the identity function, or a truncation function).[2] Additionally we require two dedicated sets $\mathcal{IV} \subset \{0,1\}^*$ and $\mathcal{C} \subset \{0,1\}^*$ the first containing *initialization vectors* and the second containing *constants*. Values from the first set allow to specify, what we will later call *initial queries*, that is, they can be used at the outset of the execution graph. The latter set $\mathcal{C}$ contains constants that, together with interim values, appear inside an execution graph. A good example where we need this distinction is the NMAC construction (see Figure 2).

We give a formal definition of the graph structure for execution graphs in Appendix A.1 and give here only a quick overview. Execution graphs consist of the following node types: $\mathcal{IV}$-nodes, $\mathcal{C}$-nodes, message-nodes, $\mathtt{h}$-nodes, $\mathtt{mp}$, $\mathtt{hp}$, and $\mathtt{hmp}$-nodes and a single $\mathtt{g}$-node. For each message block $m_1 \| \ldots \| m_\ell := \mathtt{pad}(M)$ the graph contains exactly one message-node. All outgoing edges must again be connected to a node, except for the single outgoing edge of the single $\mathtt{g}$-node. An $\mathtt{h}$-node always has two incoming edges one from an $\mathtt{hp}$-node and one from either an $\mathtt{mp}$ or an $\mathtt{hmp}$-node. Message nodes can be connected

[2] We stress that $\mathtt{g}$ is efficiently computable and not an independent (ideal) compression function as, for example, in NMAC-like functions [7].

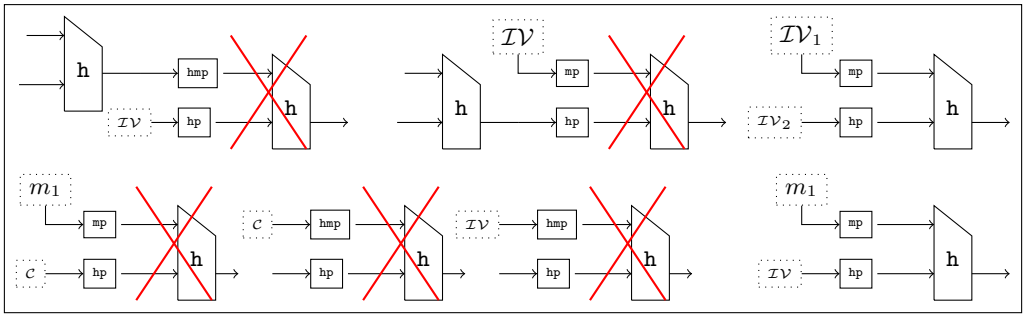

Figure 3: Restrictions on h-nodes as defined in the model. In the first case (left to right, top to bottom) an $\mathcal{IV}$-node is connected to an h-node during the computation. In the second case an $\mathcal{IV}$ node is going into a mp-node while the corresponding hp-node is not connected to an initialization vector. In the first case of the bottom row a $\mathcal{C}$-node is used together with a message-block-node. The following two cases use a $\mathcal{C}$-node (resp. an $\mathcal{IV}$-node) as input to a hmp-node. The two examples on the right show the two possible $\mathcal{IV}$ modes: the top showing *full-$\mathcal{IV}$-mode* the bottom example *single-$\mathcal{IV}$-mode*.

to mp-nodes. The outbound edges from h can be connected to either hp or hmp-nodes.[3]

We distinguish between two types of graphs: *single-$\mathcal{IV}$-mode* and *full-$\mathcal{IV}$-mode*. For execution graphs in single-$\mathcal{IV}$-mode all $\mathcal{IV}$-nodes must be connected to hp-nodes. For graphs in full-$\mathcal{IV}$-mode it must hold that if an h-node is connected to an $\mathcal{IV}$-node via an hp-node, then it must also be connected to an $\mathcal{IV}$-node via an mp-node. NMAC (Figure 2) is an example of single-$\mathcal{IV}$-mode. HMAC (Figure 10 on page 25) is an example of full-$\mathcal{IV}$-mode.

A *valid execution graph* is a non-empty graph that complies with the above rules. For each message $M \in \{0,1\}^*$ there is exactly one valid execution graph. We also use the concept of *partial execution graphs* which are non-empty graphs that comply to the above rules with the only exception that they do not contain a g-node. Hence, a partial execution graph must contain exactly one unbound outgoing edge from an h-node.

We define EVAL to be a generic, deterministic algorithm evaluating execution graphs relative to an oracle h. We here present a slightly simplified version of EVAL and give the complete version along with its pseudo-code in Appendix A.1. Let $\mathfrak{eg}$ be a valid execution graph for some message $M \in \{0,1\}^*$. To evaluate $\mathfrak{eg}$ relative to oracle h, algorithm $\texttt{EVAL}^{\texttt{h}}(\mathfrak{eg})$ performs the following steps: search for a node that has no inbound edges or for which all inbound edges are labeled. If the node is a function-node then evaluate the corresponding function using the labels from the inbound edges as input. If the node is a value-node, use the corresponding label as result. Remove the node from the graph and label all outgoing edges with the result. If the last node in the graph was removed stop and return the result. Note that $\texttt{EVAL}^{\texttt{h}}(\mathfrak{eg})$ runs in time at most $\mathcal{O}\left(|V^2|\right)$ assuming that $\mathfrak{eg}$ contains $|V|$ many nodes. If $\mathfrak{pg}$ is a partial execution graph then $\texttt{EVAL}^{\texttt{h}}(\mathfrak{pg})$, likewise, computes the partial graph outputting the result of the final h-node. We denote by $\texttt{g}(\mathfrak{pg})$ the corresponding execution graph where the single outbound h-edge of $\mathfrak{pg}$ is connected to a g-node. We call this the *completed* execution graph for $\mathfrak{pg}$.

We present examples of several restrictions on how execution graphs can be constructed in Figure 3.

Single-Round Iterative Hash Functions. Given the syntax of execution graphs we can now go on to define single round iterative hash functions such as Merkle-Damgård-like functions. Informally, an iterative hash function consists of the definitions of the preprocessing functions, the padding function and the final transformation $\texttt{g}(\cdot)$. Furthermore, we require (efficient) algorithms that construct execution graphs as well as parse a (partial) execution graph to recover the corresponding message.

Definition 3.1. *Let $\mathcal{IV} \subset \{0,1\}^*$ be a set of initialization vectors and $|\mathcal{IV}|$ be polynomial in the security parameter λ. Let $\mathcal{C} \subset \{0,1\}^*$ be a set of named vectors with $\mathcal{C} \cap \mathcal{IV} = \emptyset$ and with $|\mathcal{C}|$ polynomial in the security parameter λ.*[4] *We say $H^{\texttt{h}}_{\texttt{g},\texttt{mp},\texttt{hp},\texttt{hmp},\texttt{pad}} : \{0,1\}^* \to \{0,1\}^n$ is a (t_c, t_e)-iterative hash function if there exist deterministic algorithms* construct *and* extract *as follows:*

[3]The difference between hp and hmp is that hpoutputs values in $\{0,1\}^k$ which hmp outputs values in $\{0,1\}^d$. Note that function h is defined as $\texttt{h} : \{0,1\}^d \times \{0,1\}^k \to \{0,1\}^s$.

[4]Sometimes the initialization vector is regarded as part of the key. This we could model such that instead of defining sets $\mathcal{IV}$ and $\mathcal{C}$ we define a joint distribution on $\{0,1\}^* \times \{0,1\}^*$ and sets $\mathcal{IV}$ and $\mathcal{C}$ are then sampled together with ideal function h.

construct: *On input* $\texttt{pad}(M)$ *for any* $M \in \{0,1\}^*$ *algorithm* `construct` *generates a valid execution graph* $\mathfrak{eg}$ *for which* $H^{\texttt{h}}_{\texttt{g,mp,hp,hmp,pad}}(M) = \texttt{EVAL}^{\texttt{h}}(\mathfrak{eg})$. *Graph* $\mathfrak{eg}$ *as constructed by* $\texttt{construct}(M)$ *contains a message-block-node for value* m *if and only if* m *is a* d*-bit block in* $\texttt{pad}(M)$. *Algorithm* `construct` *runs in time* t_c. *The graph* $\texttt{construct}(M)$ *is either of type* single-$\mathcal{IV}$-mode *or of type* multi-$\mathcal{IV}$-mode *for all messages* $M \in \{0,1\}^*$ *(cf. description on page 8).*

extract: *On input a valid execution graph* $\mathfrak{eg}$*, algorithm* `extract` *outputs message* $M \in \{0,1\}^*$ *if, and only if,* $\texttt{construct}(\texttt{pad}(M))$ *is isomorphic to* $\mathfrak{eg}$. *On input a partial execution graph* $\mathfrak{pg}$*, algorithm* `extract` *outputs message* $M \in \{0,1\}^*$ *if, and only if, the completed execution graph* $\texttt{g}(\mathfrak{pg})$ *is a valid execution graph and isomorphic to* $\texttt{construct}(\texttt{pad}(M))$. *Otherwise* `extract` *outputs* $\perp$. *Algorithm* `extract` *runs in time* t_e.

When functions $\texttt{g}, \texttt{mp}, \texttt{hp}, \texttt{hmp}$ *and* `pad` *are clear from context we simply write* $H^{\texttt{h}}$.

We provide several examples of hash constructions that are covered by Definition 3.1 in Appendix A.2. Note that neither `construct` nor `extract` gets access to the underlying function `h`. Also note that by definition of algorithm `extract` there cannot be two distinct valid execution graphs for the same message M; if $\texttt{extract}(\mathfrak{pg}) = M$ then $\mathfrak{pg}$ or $\texttt{g}(\mathfrak{pg})$ is isomorphic to $\texttt{construct}(\texttt{pad}(M))$.

MULTI-ROUND ITERATIVE HASH FUNCTIONS. Most hash functions only make a single pass over the message to compute the hash value. As we will see, multiple message-passes (or rounds, as we call them) can, however, make a hash function *stronger*. A good example of such a multi-round hash function is Liskov's Zipper Hash [23] and we have depicted the corresponding execution graph in Figure 4. Zipper Hash can be regarded as a two pass Merkle-Damgård construction where message blocks are first processed in natural order and then, additionally, in reversed order. For such multi-round iterative hash functions we extend our model of execution graphs to include special round-nodes that partition the computation into multiple rounds. In each round, each message block m_i must be processed by an h-node. Furthermore, the output of round_i must be processed by an h-node in the next round $i+1$.

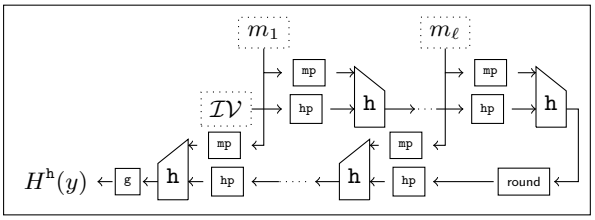

Figure 4: Zipper Hash in accordance to Definition 3.2

round-node There exist $r-1$ round-nodes with in-degree 1 taking an h-edge and out-degree 1. The outgoing edge is of type h-edge copying the label of the ingoing edge to the label of the single outbound edge. round-nodes partition the graph into distinct subgraphs and edges may not connect mp-nodes, hp-nodes, hmp-nodes or h-nodes in different subgraphs. We call the subgraph before the first round-node first round graph, the subgraph between the i-th and $i+1$-st round-node the $i+1$-st round graph and the subgraph after the $r-1$-st round-node the r-th round graph.

g-node The single g-node must be in the r-th round graph.

message-node For every message block m_i (for $1 \le i \le \ell$) there exists a node with in-degree 0 and out-degree at least r. For each message-node and round graph i there must be at least a single outbound m-edge connecting a mp node in the i-th round graph. Outgoing edges are of type m-edge and labeled with value $\langle j \rangle_{\lceil \log r \rceil} \,\|\, \langle i \rangle_{\lceil \log \ell \rceil} \,\|\, m_i$ assuming the edge goes into the j-th round graph.

With this extended definition of execution graphs we can now define a notion of multi-round iterative hash functions.

Definition 3.2. *Let the setup be as in the previous Definition 3.1. We call iterative hash function* $H^{\texttt{h}}_{r,\texttt{g,mp,hp,hmp,pad}} : \{0,1\}^* \to \{0,1\}^n$ *an* r*-round* (t_c, t_e)*-iterative hash function, if corresponding algorithm* `construct` *generates execution graphs containing* $r-1$ round*-nodes.*

When functions $\texttt{g}, \texttt{mp}, \texttt{hp}, \texttt{hmp}$ and `pad` are clear from context we simply write $H^{\texttt{h}}_r$. Note that, this definition naturally extends the previous definition as the two are equivalent for $r = 1$.

As we will later see, multi-round iterative hash functions are in some sense *stronger* than single-round constructions. The proof-of-storage example discussed in the introduction, for example, can be proven

secure for *any* multi-round indifferentiable hash constructions (such as Zipper Hash) while we have seen that it is not secure for various single-round hash constructions, even in case they are indifferentiable from a random oracle.

3.1 Important h-Queries

Considering the execution of hash functions as graphs allows us to identify certain types of "important" queries by their position in the graph relative to a function $\mathtt{h}$. Assume that $Q = (m_i, x_i)_{1 \le i \le p}$ is an ordered sequence of $\mathtt{h}$-queries to compression function. If we consider the i-th query $q_i = (m_i, x_i)$ then only queries appearing before q_i in Q are relevant for our upcoming naming conventions. For initial queries we distinguish between the two $\mathcal{IV}$-modes: single and full (see also description of $\mathtt{h}$-node and Figure 3). For single-$\mathcal{IV}$-mode, we call q_i an *initial query* if, and only if, $\mathtt{hp}^{-1}(x_i) \in \mathcal{IV}$. For full-$\mathcal{IV}$-mode, we call q_i an *initial query* if, and only if, in addition to $\mathtt{hp}^{-1}(x_i) \in \mathcal{IV}$ it also holds that $\mathtt{mp}^{-1}(m_i) \in \mathcal{IV}$.

Besides initial queries we are interested in queries that occur "in the execution graph" and we call these *chained queries*. We call query q_i a *chained query* if given the queries appearing before q_i there exists a valid (partial) execution graph containing an h-node with inbound edges transporting values m_i and x_i. Finally, we call query q_i result query for message M, if $\mathtt{g}(q_i) = H^{\mathtt{h}}(M)$ and q_i is a chained query.

Definition 3.3. *Let $Q = (m_i, x_i)_{1 \le i \le p}$ be a sequence of queries to $\mathtt{h} : \{0,1\}^d \times \{0,1\}^k \to \{0,1\}^s$. Let $q_i = (m_i, x_i)$ be the i-th query in Q. For single-$\mathcal{IV}$-mode let the predicate $\mathsf{init}^Q(q_i) := \mathsf{init}^Q(m_i, x_i)$ be true if, and only if,*

$$\mathtt{hp}^{-1}(x_i) \in \mathcal{IV} \ .$$

For full-$\mathcal{IV}$-mode let the predicate $\mathsf{init}^Q(q_i)$ be true if, and only if,

$$\mathtt{hp}^{-1}(x_i) \in \mathcal{IV} \quad \wedge \quad \mathtt{mp}^{-1}(m_i) \in \mathcal{IV} \ .$$

We define the predicate $\mathsf{chained}^Q(m_i, x_i)$ to be true if, and only if,

$$\mathsf{init}(m_i, x_i) \quad \vee \quad \exists\ j \in [i-1] : (\mathsf{chained}(q_j) \wedge \mathtt{hp}(\mathtt{h}(q_j)) = x_i) \ .$$

We define the predicate $\mathsf{result}^{Q,M}(m_i, x_i)$ to be true if, and only if,

$$\mathsf{chained}^Q(m_i, x_i) \quad \wedge \quad H^{\mathtt{h}}(M) = \mathtt{g}(\mathtt{h}(m_i, x_i)) \ .$$

We drop the reference to the query set Q if it is clear from context.

In other words: query q_i is a chaind query if either q_i is an initial query, or there exists a query q_j that came before q_i which was a chained query and for which $\mathtt{hp}(h(q_j)) = \mathtt{hp}(h(m_j, x_j)) = x_i$. Thus, if $\mathtt{hp}(\cdot)$ is the identity function, as it is for example in chop-MD, or NMAC, we require that value x_i was generated by an h-query. Also, let us stress, that the predicates hold, or do not hold, relative to the previous queries given by sequence Q and are not affected by later queries. We depict the different query types in Figure 5.

3.2 A Missing Link

In the following we show that, if for some message M with corresponding execution graph $\mathtt{pg} \leftarrow \mathtt{construct}(M)$ an adversary does not make all $\mathtt{h}$-queries in $\mathtt{EVAL}^{\mathtt{h}}(\mathtt{pg})$ then the probability of the adversary computing $H^{\mathtt{h}}(M)$ is low. This is formally captured by the following lemma:

Lemma 3.4. *Let function $H^{\mathtt{h}} : \{0,1\}^* \to \{0,1\}^n$ be an iterative hash function and let $\mathtt{h} : \{0,1\}^d \times \{0,1\}^k \to \{0,1\}^s$ be a fixed-length random oracle. Let $\mathcal{A}^{\mathtt{h}}$ be an adversary that makes at most $q_{\mathcal{A}}$ many queries to $\mathtt{h}$. Let $Q = (m_i, x_i)_{1 \le i \le q_{\mathcal{A}}}$ denote the queries to $\mathtt{h}$ on an $\mathcal{A}^{\mathtt{h}}(1^\lambda)$ invocation. Let $\mathsf{qry}^{\mathtt{h}}(H^{\mathtt{h}}(M))$ denote the set of $\mathtt{h}$-queries during the computation of $H^{\mathtt{h}}(M)$ for message $M \in \{0,1\}^*$. Then it holds that*

$$\Pr\left[\,(M,y) \leftarrow \mathcal{A}^{\mathtt{h}}(1^\lambda) : H^{\mathtt{h}}(M) = y \wedge \exists (m,x) \in \mathsf{qry}^{\mathtt{h}}(H^{\mathtt{h}}(M))\ s.t\ (m,x) \notin Q\,\right] \le \frac{q_{\mathcal{A}}}{2^s} + \frac{1}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}}$$

where U_s denotes a random variable uniformly distributed in $\{0,1\}^s$. The probability is over the choice of random oracle $\mathtt{h}$ and the coins of $\mathcal{A}$.

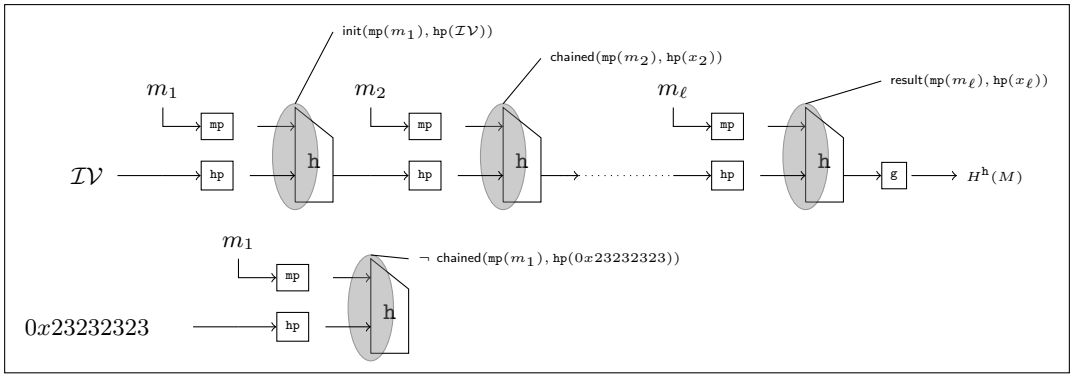

Figure 5: Denoting queries in the Merkle-Damgård construction where we assume that $0x23232323$ is not a valid initialization vector. Furthermore: $x_2 := \mathtt{h}(\mathtt{mp}(m_1), \mathtt{hp}(\mathcal{IV}))$ and value x_l is computed recursively as $x_\ell := \mathtt{h}(\mathtt{mp}(m_\ell), \mathtt{hp}(x_{\ell-1}))$.

Proof. Assume adversary $\mathcal{A}$ succeeds, that is, it outputs a message M and value y such that $H^{\mathtt{h}}(M) = y$ and there exists an $\mathtt{h}$-query (m, x) which occurs during the evaluation of $H^{\mathtt{h}}(M)$ but which was not queried by $\mathcal{A}$. We consider the execution graph $\mathtt{pg} \leftarrow \mathtt{construct}(M)$ which induces a partial order on its nodes. That is, if n_1 and n_2 are nodes in $\mathtt{pg}$ then we write $n_2 \succ n_1$ if, and only if, there exists a directed path from node n_1 to node n_2 in $\mathtt{pg}$. Relative to oracle $\mathtt{h}$ we can identify an $\mathtt{h}$-node in graph $\mathtt{pg}$ for which the two input edges transport values m and x. We call this node $\mathtt{node}_{m,x}$ and in case there are multiple choices we simply choose one at random. As $\mathcal{A}$ does not query $\mathtt{h}(m, x)$ it holds that this value has min-entropy s-bits, that is, $\mathrm{H}_\infty(\mathtt{h}(m, x)) = s$. Value $\mathtt{h}(m, x)$ is transported on all outgoing edges from $\mathtt{node}_{m,x}$. Each of these edges is connected to an $\mathtt{hp}$ or an $\mathtt{hmp}$-edge. By definition we have that

$$\mathrm{H}_\infty(\mathtt{hp}(U_s)) = \mathrm{H}_\infty(\mathtt{hmp}(U_s)) = s$$

where U_s is a random variable uniformly distributed in U_s. In other words, as preprocessing functions $\mathtt{hp}$ and $\mathtt{hmp}$ are injective they do not decrease entropy. Thus the sole outgoing edge of the preprocessing node transports s-bits of min-entropy to an $\mathtt{h}$-node $\mathtt{node}^*$ for which $\mathtt{node}^* \succ \mathtt{node}_{m,x}$.

Let $\mathtt{node}_{\mathtt{res}}$ denote the final $\mathtt{h}$-node in graph $\mathtt{pg}$. As for any $\mathtt{h}$-node $\mathtt{node}'$ in $\mathtt{pg}$ it holds

$$\mathtt{node}' = \mathtt{node}_{\mathtt{res}} \quad \vee \quad \mathtt{node}_{\mathtt{res}} \succ \mathtt{node}'$$

we get by recursively repeating the above argument that one of the input edges to $\mathtt{node}_{\mathtt{res}}$ transports s bits of min-entropy. Let $(m_{\mathtt{res}}, x_{\mathtt{res}})$ be the values transported on the two edges going into the final $\mathtt{h}$-node $\mathtt{node}_{\mathtt{res}}$. Then, we have that the probability that $\mathcal{A}$ queries $\mathtt{h}$ on $(m_{\mathtt{res}}, x_{\mathtt{res}})$ if it did not query $\mathtt{h}(m, x)$ is upper bounded by $q_{\mathcal{A}} \cdot 2^{-s}$.

By the above discussion we also directly yield that $\mathrm{H}_\infty(\mathtt{h}(m_{\mathtt{res}}, x_{\mathtt{res}})) = s$. Thus the probability of $\mathcal{A}$ guessing value y such that $\mathtt{g}(\mathtt{h}(m_{\mathtt{res}}, x_{\mathtt{res}})) = y$ is upper bounded by $2^{-\mathrm{H}_\infty(\mathtt{g}(U_s))}$ where again U_s denotes a random variable uniformly distributed in $\{0,1\}^s$. □

3.3 Extracting the Execution Graph

Given a message $M \in \{0,1\}^*$ the corresponding execution graph $H^{\mathtt{h}}(M)$ can be (efficiently) generated given algorithm $\mathtt{construct}$. In the following we show that assuming that $\mathtt{h}$ is a fixed length random-oracle —that is a function chosen uniformly from the space of all functions of the form $\{0,1\}^d \times \{0,1\}^k \to \{0,1\}^s$— then, given a sequence $Q = (m_i, x_i; y_i)_{1 \le i \le p}$ of queries (and corresponding answers) to $\mathtt{h}$ we can efficiently construct all possible corresponding valid execution graphs for a given construction $H^{\mathtt{h}}$. More formally, we show that there exists an efficient algorithm (called extractor $\mathcal{E}$) that given a sequence of $\mathtt{h}$-queries and corresponding answers can output a target set of messages $\mathcal{M}$ such that no, even unbounded adversary, that is given query-set Q and corresponding answers can output a tuple M, y such that $H^{\mathtt{h}}(M) = y$ and $M \notin \mathcal{M}$. This is formally captured by the following lemma.

Lemma 3.5. *Let function $H^{\mathtt{h}} : \{0,1\}^* \to \{0,1\}^n$ be a (t_c, t_e)-iterative hash function and let $\mathtt{h} : \{0,1\}^d \times \{0,1\}^k \to \{0,1\}^s$ be a fixed-length random oracle. Let $\mathcal{A}^{\mathtt{h}}$ be an adversary that makes at most $q_{\mathcal{A}}$ many queries to $\mathtt{h}$. Let $Q = (m_i, x_i; y_i)_{1 \le i \le q_{\mathcal{A}}}$ denote the queries to $\mathtt{h}$ together with the corresponding answers*

$y_i := \mathtt{h}(m_i, x_i)$ *on an* $\mathcal{A}^{\mathtt{h}}(1^\lambda)$ *invocation. Then there exists an extractor* $\mathcal{E}$*, running in time at most* $\mathcal{O}(q_{\mathcal{A}} \cdot t_e)$ *and outputting a set* $\mathcal{M}$ *with* $|\mathcal{M}| \leq 2q_{\mathcal{A}}$*, such that*

$$\Pr\left[(M, y) \leftarrow \mathcal{A}^{\mathtt{h}}(1^\lambda); \mathcal{M} \leftarrow \mathcal{E}(Q) : H^{\mathtt{h}}(M) = y \wedge M \notin \mathcal{M}\right] \leq \frac{q_{\mathcal{A}} + 3q_{\mathcal{A}}^2}{2^s} + \frac{1}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}} \ .$$

The probability is over the coins of $\mathcal{A}^{\mathtt{h}}$ *and the choice of random oracle* $\mathtt{h}$*. Value* U_s *denotes a random variable uniformly distributed in* $\{0,1\}^s$*.*

Proof. We will first present extractor $\mathcal{E}$ (see Figure 6) to then argue that it achieves the claimed bound. Extractor $\mathcal{E}$ will work with partial graphs that we will store in a set $\mathcal{PG}$. Without loss of generalization we assume that Q does not contain the same query twice, that is $\mathcal{A}$ does repeat queries to $\mathtt{h}$.

The extractor will do a single pass over query sequence Q. In each step extractor $\mathcal{E}$ initializes new partial graphs if the current query is an initial query. A freshly initialized graph consists of either an $\mathcal{IV}$-node connected to a $\mathtt{hp}$-node, a message-block-node (or a second $\mathcal{IV}$-node, depending on the $\mathcal{IV}$-mode) connected to a $\mathtt{mp}$-node and an $\mathtt{h}$-node which is connected to the $\mathtt{mp}$ and $\mathtt{hp}$ nodes. The outgoing $\mathtt{h}$-edge is free. We denote the creation of new partial graphs for query (m, x) by

$$\textbf{new}\ \mathrm{PartialGraph}(\mathtt{mp}^{-1}(m), \mathtt{hp}^{-1}(x))$$

Whenever a new partial graph is constructed (or later extended) we compute the value of the sole outgoing $\mathtt{h}$-edge and denote it by $\mathtt{pg.y} \leftarrow y$. Note that, by construction we have that $\mathtt{pg.y} = y = \mathtt{EVAL}^{\mathtt{h}}(\mathtt{pg})$.

If query $(m, x, y) \in Q$ is not an initial query we try to extend existing partial graphs in $\mathcal{PG}$ by this query. A partial graph $\mathtt{pg} \in \mathcal{PG}$ can be extended if its sole outgoing $\mathtt{h}$-edge which transports value $\mathtt{pg.y}$ can be connected to a new $\mathtt{h}$-node constructed from query (m, x). This is the case, if and only if, (i) $\mathtt{pg.y} = \mathtt{hp}^{-1}(x)$ or if (ii) $\mathtt{pg.y} = \mathtt{hmp}^{-1}(m)$. Note that for the second case, where $\mathtt{pg.y} = \mathtt{hmp}^{-1}(m)$ we can only extend the partial graph if there is also $\mathtt{pg}' \in \mathcal{PG}$ such that $\mathtt{pg}'.\mathtt{y} = \mathtt{hp}^{-1}(x)$ as otherwise the $\mathtt{hp}$-node would not have all its input edges bound, thus the new partial graph is constructed from two previously existing graphs.[5] If we extend a partial graph, a new partial graph is generated for the extended graph and the old one is kept in $\mathcal{PG}$. We denote by

$$\mathtt{pg}.\mathrm{extendedBy}(m, x, y)$$

the partial graph generated from $\mathtt{pg}$ and extended by query $(m, x, y) \in Q$ (corresponding to case (i), see above) and by

$$\mathtt{pg}.\mathrm{extendedBy}(\mathtt{pg}', m, x, y)$$

the partial graph generated from the two partial graphs $\mathtt{pg}$ and $\mathtt{pg}'$ extended by query $(m, x, y) \in Q$ which corresponds to case (ii). Again, after a new graph is constructed we set $\mathtt{pg.y} \leftarrow y$. Note that also here, by construction, we have that $\mathtt{pg.y} = y = \mathtt{EVAL}^{\mathtt{h}}(\mathtt{pg})$.

After all partial graphs are constructed the extractor then recovers for each partial graph the sequence of message-blocks using algorithm `extract`. These, form the set of target messages output by extractor $\mathcal{E}$. We give the pseudo-code for extractor $\mathcal{E}$ in Figure 6.

It remains to argue that extractor $\mathcal{E}$ has the claimed runtime, as well as that the target set $\mathcal{M}$ as output by $\mathcal{E}$ is sufficient. For the run-time note that the extractor makes a single pass over the query set Q. If adversary $\mathcal{A}$ did not find collisions in $\mathtt{h}$ (which only happens with probability less than $q_{\mathcal{A}}^2 \cdot 2^{-s+1}$) in each step at most 2 new partial graphs are generated. Thus, with overwhelming probability the number of generated partial graphs is at most $2|Q| = 2q_{\mathcal{A}}$ (for the case that more than $2q_{\mathcal{A}}$ partial graphs are found we assume that $\mathcal{E}$ stops and the adversary wins which corresponds to parts of the first term in the statement of Lemma 3.5). For each of the partial graphs we run `extract` once, which leaves us with a runtime of $\mathcal{O}(2q_{\mathcal{A}} \cdot t_e)$ where t_e denotes the run-time of deterministic algorithm `extract`.

Let us now show that $\mathcal{M}$ is sufficient. By Lemma 3.4 we have that all $\mathtt{h}$-queries occurring during the computation of $H^{\mathtt{h}}(M)$ must be in Q but for probability

$$\frac{q_{\mathcal{A}}}{2^s} + \frac{1}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}}$$

[5] See the HMAC or NMAC construction (Appendix A.2) for an example, where this case can occur.

Extractor: $\mathcal{E}(Q)$
$\mathcal{M} \leftarrow \{\}; \mathcal{PG} \leftarrow \{\}$
for $i = 1 \ldots |Q|$ **do** /* building partial graphs */
$(m, x, y) \leftarrow Q[i]$
if $\mathsf{init}(m, x)$ **then**
$\mathfrak{newG} \leftarrow$ **new** PartialGraph$(\mathtt{mp}^{-1}(m), \mathtt{hp}^{-1}(x))$
$\mathfrak{newG}.\mathsf{y} \leftarrow y$
$\mathcal{PG}$.add$(\mathfrak{newG})$
else
foreach $\mathfrak{pg} \in \mathcal{PG} : \mathfrak{pg}.\mathsf{y} = \mathtt{hp}^{-1}(x)$ **do**
$\mathfrak{newG} \leftarrow \mathfrak{pg}$.extendedBy$(m, x, y)$
$\mathfrak{newG}.\mathsf{y} \leftarrow y$
$\mathcal{PG}$.add$(\mathfrak{newG})$
foreach $(\mathfrak{pg}, \mathfrak{pg}') \in \mathcal{PG} \times \mathcal{PG} : \mathfrak{pg}.\mathsf{y} = \mathtt{hmp}^{-1}(m) \wedge \mathfrak{pg}'.\mathsf{y} = \mathtt{hp}^{-1}(x)$ **do**
$\mathfrak{newG} \leftarrow \mathfrak{pg}$.extendedBy$(\mathfrak{pg}', m, x, y)$
$\mathfrak{newG}.\mathsf{y} \leftarrow y$
$\mathcal{PG}$.add$(\mathfrak{newG})$
foreach $\mathfrak{pg} \in \mathcal{PG}$ **do** /* building target message set */
$M \leftarrow \mathtt{extract}(\mathfrak{pg})$
if $M \neq \perp$ **do**
$\mathcal{M} \leftarrow \mathcal{M} \cup M$
return $\mathcal{M}$

Figure 6: The extractor for Lemma 3.5.

We now show, that the queries appear in the correct order but for small probability. The execution graph $\mathfrak{pg} \leftarrow \mathtt{construct}(M)$ induces a partial order on its nodes. That is, if n_1 and n_2 are nodes in $\mathfrak{pg}$ we write $n_2 \succ n_1$ if, and only if, there exists a directed path from node n_1 to node n_2. Relative to $\mathtt{h}$ and message M we can identify each $\mathtt{h}$-node in $\mathfrak{pg}$ by the values transported on the two ingoing edges m and x: we write $\mathtt{node}_{m,x}$. This, identification is unique, unless adversary $\mathcal{A}$ finds a collision on $\mathtt{h}$ which happens at most with probability

$$\frac{q_{\mathcal{A}}}{2^{s-1}} \ .$$

Let nodes $\mathtt{node}_{m,x}$ and $\mathtt{node}_{m',x'}$ be two neighbored nodes in $\mathfrak{pg}$ such that $\mathtt{node}_{m',x'} \succ \mathtt{node}_{m,x}$. By neighbored we mean that on the path from $\mathtt{node}_{m,x}$ to $\mathtt{node}_{m',x'}$ no other $\mathtt{h}$-nodes are passed. Clearly $q_{\mathcal{A}}$ is an upper bound on the number of neighbored pairs. We are interested in the probability, that although node $\mathtt{node}_{m,x}$ comes before node $\mathtt{node}_{m',x'}$ in $\mathfrak{pg}$ the corresponding queries in Q are in reverse order, that is, there query (m, x) appears only after query (m', x'). This probability is upper bounded by $q_{\mathcal{A}} \cdot 2^{-s}$, since s-bits of min-entropy are transported from node $\mathtt{node}_{m,x}$ to node $\mathtt{node}_{m',x'}$ (also see Lemma 3.4) and the adversary had at most $q_{\mathcal{A}}$ tries to guess correctly. Thus, we can upper bound the probability, that there two queries in Q corresponding to neighboring $\mathtt{h}$-nodes in the execution graph of $\mathfrak{pg}$ and are out of order by

$$\frac{q_{\mathcal{A}}^2}{2^s}$$

Putting it all together, we have that if $\mathcal{A}$ does not find any collision on $\mathtt{h}$ then for any M output by $\mathcal{A}$ all $\mathtt{h}$-queries in $H^{\mathtt{h}}(M)$ must be in Q but for the guessing probability

$$\frac{q_{\mathcal{A}}}{2^s} + \frac{1}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}} \ .$$

Furthermore, the queries must appear in topologically correct order but for probability $q_{\mathcal{A}}^2 \cdot 2^{-s}$. Then, however, the corresponding message M is reconstructed also by $\mathcal{E}$ as by definition it reconstructs partial graphs if all queries appear in topologically correct order. □

3.4 h-Queries during Functionality Respecting Games

As Ristenpart et al. [31], we only consider the class of *functionality-respecting* games (see Section 2.3) where only adversarial procedures may call the adversarial interface of functionalities (i.e., the underlying function $\mathtt{h}$ in our case). We now define various terms that allow us to talk about specific queries from adversarial procedures to the underlying function $\mathtt{h}$ of iterative hash function $H^{\mathtt{h}}$ during game G.

Definition 3.6. *Let $G^{H^{\mathtt{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m}$ be a functionality respecting game with access to hash functionality $H^{\mathtt{h}}$ and adversarial procedures $\mathcal{A}_1,\ldots,\mathcal{A}_m$. We denote by $\mathsf{qry}^{G,\mathtt{h}}$ the sequence of queries to the adversarial interface of $H^{\mathtt{h}}$ (that is, $\mathtt{h}$) during the execution of game G.*

Note that $\mathsf{qry}^{G,\mathtt{h}}$ is a random variable over the random coins of game G. Thus, we can regard the query sequence as a deterministic function of the random coins. In this light, in the following we define subsequences of queries belonging to certain adversarial procedures such as the i-th query of the j-th adversarial procedure.

Game $G^{H^{\mathtt{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m}$ can call adversarial procedures $\mathcal{A}_1,\ldots,\mathcal{A}_m$ in any order and multiple times. That is, the 3rd call by G's main method might be to procedure $\mathcal{A}_1$. Thus, we first define a mapping from the sequence of adversarial procedure calls by the game's main procedure to the actual adversarial procedure $\mathcal{A}_i$. For better readability, we drop the superscript identifying game G in the following definitions and whenever the game is clear from context. Similarly, we drop the superscript identifying oracle $\mathtt{h}$ exposed by the adversarial interface of functionality $H^{\mathtt{h}}$ if clear from context.

Definition 3.7. *We define AdvSeq_i (for $i \geq 1$) to denote the adversarial procedure corresponding to the i-th adversarial procedure call by game G. We set $|\mathsf{AdvSeq}|$ to denote the total number of adversarial procedure calls by G.*

We now define the sequence of $\mathtt{h}$-queries that occur during the execution of the i-th adversarial procedure AdvSeq_i.

Definition 3.8. *By qry_i we denote the sequence of queries to $\mathtt{h}$ by procedure AdvSeq_i during the i-th adversarial procedure call by the game's main procedure. By $\mathsf{qry}_{i,j}$ we denote the j-th query in this sequence.*

We also need a notion which captures all those queries executed before a specific adversarial procedure AdvSeq_i was called. For this, we will slightly abuse notation and "concatenate" two (or more) sequences, i.e., if S_1 and S_2 are two sequences, then by $S_1||S_2$ we denote the sequence that contains all elements of S_1 followed by all elements of S_2 in their specific order.

Definition 3.9. *By $\mathsf{qry}_{<i}$ we denote the sequence of queries to procedure $\mathtt{h}$ before the execution of procedure AdvSeq_i. By $\mathsf{qry}_{<i,j}$ we denote the sequence of queries to procedure $\mathtt{h}$ up to the j-th query of the i-th adversarial procedure call. Formally,*

$$\mathsf{qry}_{<i} := \Big|\Big|_{k=1}^{i-1} \mathsf{qry}_k \qquad \textit{and} \qquad \mathsf{qry}_{<i,j} := \mathsf{qry}_{<i} \,||\, \Big|\Big|_{k=1}^{j-1} \mathsf{qry}_{i,k}$$

Finally, we define the sequence of $\mathtt{h}$-queries by procedure AdvSeq_i up-to the i-th adversarial procedure call by the game's main procedure. That is, in addition to queries qry_i we have all queries from previous calls to AdvSeq_i by the game's main procedure.

Definition 3.10. *By $\mathsf{qry}_{<\mathcal{A}_i,j}$ we denote the sequence of queries to procedure $\mathtt{h}$ by the i-th adversarial procedure AdvSeq_i up-to query $\mathsf{qry}_{<i,j}$. Formally,*

$$\mathsf{qry}_{<\mathcal{A}_i,j} := \Big|\Big|_{\substack{0<\ell<i,\\ \mathsf{AdvSeq}_\ell=\mathsf{AdvSeq}_i}} \mathsf{qry}_\ell \;\;\|\;\; \Big|\Big|_{k=1}^{j-1} \mathsf{qry}_{i,k} \;.$$

BAD QUERIES. Having defined queries to the adversarial interface of the hash functionality (i.e., underlying function $\mathtt{h}$) occurring during a game G allows us to use our notation established in Section 3.1 on $\mathtt{h}$-queries: initial queries, chained queries and result queries, for example, saying query $\mathsf{qry}_{i,j}$ is an initial query. In addition, we now define a bad event corresponding to $\mathtt{h}$-queries.

In multi-stage protocols an adversary can split the computation of a hash value over several adversarial procedures by passing on intermediate values resulting from the $\mathtt{h}$-computations by $H^{\mathtt{h}}$ (think of Ristenpart et al.'s proof-of storage example in the introduction). Further note that when we try to apply the idea behind the MRH theorem to multi-stage games, this poses a problem to simulators (also see Section 2.2). As we will later see this is, in fact, the only problem why the MRH theorem cannot be applied directly. We will label such queries *bad queries*.

Informally, we call a query (m, x) to function $\mathtt{h}(\cdot,\cdot)$ bad if it is a chained query with respect to all previous queries during the game, but it is not a chained query if we restrict the sequence of queries to that of the current adversarial procedure. Note that, whether or not a query is bad only depends on queries to $\mathtt{h}$ prior to the query and is not changed by any query coming later in the game. (Note the change in the underlying sequence for the two predicates in the following definition.)

Definition 3.11. *Let $G^{H^{\mathtt{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m}$ be any game. Let $(m, x) := \mathsf{qry}_{i,j}$ be the j-th query to function $\mathtt{h}$ by adversary AdvSeq_i. Then query (m, x) is called $\mathsf{bad}^{\mathcal{A}_i}(\mathsf{qry}_{i,j})$ if, and only if:*

$$\mathsf{chained}^{\mathsf{qry}_{<i,j}}(m, x) \quad \textit{and} \quad \neg\mathsf{chained}^{\mathsf{qry}_{<\mathcal{A}_i,j}}(m, x)$$

4 Unsplittable Multi-stage Games

We now present Definition 4.1, the main definition of this paper. We call a game $G \in \mathcal{LG}$ *unsplittable* for an iterative hash construction $H^{\mathtt{h}}$, if two conditions hold:

- For any adversary $\mathcal{A}_1, \ldots, \mathcal{A}_m$ there exists adversary $\mathcal{A}_1^*, \ldots, \mathcal{A}_m^*$ such that games $G^{H^{\mathtt{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m}$ and $G^{H^{\mathtt{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*}$ change only by a small factor.
- During game $G^{H^{\mathtt{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*}$ we have that bad queries only occur with small probability.

Intuitively, this means that it does not help adversaries to split up the computation of hash values over several distinct adversarial procedures.

In Section 5 we give a composition theorem which informally states that if a game is *unsplittable* for an indifferentiable hash construction $H^{\mathtt{h}}$, then security proofs in the random oracle carry over if the random oracle is implemented by that particular hash function. We will see, that the examples of the introduction, the CDA and the CRP game, are *unsplittable* for **any** (resp. any two-round) indifferentiable hash construction, thus, allowing us to use the MRH composition theorem to proof security in the random oracle model.

Definition 4.1. *Let $H^{\mathtt{h}}$ be an iterative hash function and let $\mathtt{h} : \{0,1\}^d \times \{0,1\}^k \to \{0,1\}^s$ be an ideal function. We say a functionality respecting game $G \in \mathcal{LG}$ is $(t_{\mathcal{A}^*}, q_{\mathcal{A}^*}, \epsilon_G, \epsilon_{\mathsf{bad}})$*-unsplittable *for $H^{\mathtt{h}}$ if for every adversary $\mathcal{A}_1, \ldots, \mathcal{A}_m$ there exists algorithm $\mathcal{A}_1^*, \ldots, \mathcal{A}_m^*$ such that for all values y*

$$\Pr\left[G^{H^{\mathtt{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m} \Rightarrow y\right] \leq \Pr\left[G^{H^{\mathtt{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*} \Rightarrow y\right] + \epsilon_G \ .$$

Adversary $\mathcal{A}_i^*$ has run-time at most $t^*_{\mathcal{A}_i}$ and makes at most $q^*_{\mathcal{A}_i}$ queries to $\mathtt{h}$. Moreover, it holds for game $G^{H^{\mathtt{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*}$ that:

$$\Pr\left[\exists i \in \{1,\ldots,|\mathsf{AdvSeq}|\}, \exists j \in \{1,\ldots,q^*_{\mathcal{A}_i}\} \ : \mathsf{bad}^{\mathcal{A}_i}(\mathsf{qry}_{i,j})\right] \leq \epsilon_{\mathsf{bad}} \ .$$

The probability is over the coins of game $G^{H^{\mathtt{h}},\mathcal{A}^*{}_1,\ldots,\mathcal{A}^*{}_m}$ and the choice of function $\mathtt{h}$.

Unsplittability is defined relative to a game G **and** a hash function $H^{\mathtt{h}}$. We say that game G is *unsplittable* for $H^{\mathtt{h}}$ if for every adversary $\mathcal{A}_1, \ldots, \mathcal{A}_m$ there exist procedures $\mathcal{A}_1^*, \ldots, \mathcal{A}_m^*$ such that the games $G^{H^{\mathtt{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m}$ and $G^{H^{\mathtt{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*}$ are ϵ_G-close, while during the game with procedures $\mathcal{A}_1^*, \ldots, \mathcal{A}_m^*$ bad queries only occur with probability at most ϵ_{bad}. Intuitively this means that bad queries should

not be necessary, or in other words, anything that can be done with bad queries, can also be achieved without resorting to bad queries.

Assuming that all algorithms run in polynomial time and that all ϵ-probabilities are negligible this means, that we can exchange a game $G^{H^{\mathtt{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m}$ for game $G^{H^{\mathtt{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*}$ such that no efficient distinguisher can detect any difference and further that bad queries during $G^{H^{\mathtt{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*}$ do only occur with negligible probability. Together with a composition theorem that we will prove in Section 5 this yields that security proofs in the random oracle model carry over for games that are *unsplittable* for an indifferentiable hash function $H^{\mathtt{h}}$, if we replace the random oracle by this hash function.

4.1 CDA and CRP are Unsplittable

We now show that chosen distribution attack (CDA) game [6] is *unsplittable* for any iterative hash function and the proof-of-storage challenge-response (CRP) game [31] is *unsplittable* for any 2-round iterative hash function.

The non-adaptive CDA game (see Figure 1) describes security for deterministic public-key encryption schemes $\mathcal{AE}$ (we give an introduction in Appendix D). To prove that the non-adaptive CDA game is *unsplittable* we need three (reasonable) assumptions about the encryption scheme: first, it should be infeasible to guess the public key as being generated by the corresponding key generation algorithm KGen. We denote the corresponding probability by $\mathsf{maxpk}_{\mathcal{AE}}$. Second, the encryption scheme must be such that no adversary can distinguish between real and simulated encryptions. This is formally captured by requiring that the encryption scheme is IND-SIM secure [31] in the ROM against adversaries that do not query the random oracle (see Appendix D.2). Finally, we assume that the encryption scheme includes the public key, the to be encrypted message, and the given randomness in the single random oracle query per $\mathcal{E}$-invocation. Let us note that this setup is equivalent to the setup in [31], where Ristenpart et al. show that the MRH composition theorem for non-adaptive CDA works for NMAC-like hash function of the form $\mathtt{g}(f^{\mathtt{h}}(\cdot))$ where $\mathtt{g}$ and $\mathtt{h}$ are two independent ideal functions. The main difference between their and our proof is our proof holds for any iterative hash function and not just for NMAC-like functions.

Lemma 4.2. *Let $\mathcal{AE}$ be a public-key encryption scheme. Let the encryption scheme query its hash construction $H^{\mathtt{h}}$ on a single message per $\mathcal{E}$ invocation, that message including (an encoding of) the public key and the input to $\mathcal{E}$. Let adversary $\mathcal{A}_1$ be a valid (μ,ν)-$\mathtt{mmr}$-source. Then, for any encryption simulator $\mathcal{S}$ there exists IND-SIM-adversary $\mathcal{B}$ such that the non-adaptive $CDA^{H^{\mathtt{h}},\mathcal{A}_1,\mathcal{A}_2}_{\mathcal{AE}}$ game (cf. Figure 1) is $(t_{\mathcal{A}^*}, q_{\mathcal{A}^*}, \epsilon_G, \epsilon_{\mathsf{bad}})$-*unsplittable *for any hash construction. Moreover,*

$$\epsilon_G \le 3q_{\mathcal{A}_1}\cdot \mathsf{maxpk}_{\mathcal{AE}} + \frac{3(q_{\mathcal{A}_1}+q_{\mathcal{A}_2})\cdot\nu + 2q_{\mathcal{B}} + 6q_{\mathcal{B}}^2}{2^s} + \frac{3q_{\mathcal{A}_2}\cdot\nu}{2^\mu} + \frac{6\nu}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}} + 2\cdot\mathbf{Adv}^{\textit{IND-SIM}}_{\mathcal{AE},\mathcal{R},\mathcal{S}}(\mathcal{B})$$

$$\epsilon_{\mathsf{bad}} = 0 \qquad t_{\mathcal{A}_i^*} \le t_{\mathcal{A}_i} \qquad q_{\mathcal{A}_i^*} \le 2q_{\mathcal{A}_i} \qquad q_{\mathcal{B}} \le q_{\mathcal{A}_1} + q_{\mathcal{A}_2} + \nu\cdot\max(q_{\mathcal{E}}, q_{\mathcal{S}})$$

Adversary $\mathcal{B}$ runs in time of game $CDA^{H^{\mathtt{h}},\mathcal{A}_1,\mathcal{A}_2}_{\mathcal{AE}}$, makes no random oracle queries and ν many $\mathtt{RoS}$*-queries. Value $t_{\mathcal{A}_i}$ (resp. $q_{\mathcal{A}_i}$) denotes the run-time of procedure $\mathcal{A}_i$ (resp. the number of oracle queries by adversary $\mathcal{A}_i$). Value U_s denotes a random variable that is uniformly distributed in $\{0,1\}^s$. Values $q_{\mathcal{E}}, q_{\mathcal{S}}$ are upper bounds on the number of (indirect)* $\mathtt{h}$*-queries during the computation of $\mathcal{E}^{H^{\mathtt{h}}}$ (resp. $\mathcal{S}^{H^{\mathtt{h}}}$).*

The main idea behind the proof is to exchange the real encryptions by simulated encryptions, thus destroying the only means of communication between the two adversaries. We give the full proof in Appendix D.2. Additionally, in Appendix D.1, we proof a second related Lemma for non-adaptive CDA under an independent set of assumptions (that is, instead of assuming the encryption scheme to be IND-SIM-secure we have an additional assumption on the hash function). We believe the proof to be quite instructive on how to work with the notions of bad queries and *unsplittable*-games.

The Adaptive CDA Game. In the adaptive case, the first adversary $\mathcal{A}_1$ is allowed to base its message and randomness vectors upon seen encryptions, that is, it is given access to an additional encryption oracle (see Figure 20 in Appendix E). In this case, we have to ensure that encryptions do not leak too much information on the public key, which can be regarded as the complement notion to $\mathsf{maxpk}_{\mathcal{AE}}$ for

the non-adaptive case. For this we introduce the notion of PK-EXT (short for *public-key extractability*) which we believe to be of independent interest. Intuitively, PK-EXT says that no efficient adversary that is given access to an encryption oracle can guess the public key used in the encryption with more than negligible probability. PK-EXT can thus be regarded as an adaption of the key indistinguishability notion by Bellare et al. [4] for the setting of deterministic encryption. We formally introduce PK-EXT (and show that it is met by existing constructions) as well as prove that, under this assumption, the adaptive CDA game is *unsplittable* for any indifferentiable hash construction in Appendix E.

THE CRP GAME. With the next lemma we establish that the challenge-response game(cf. Figure 1) from the introduction is *unsplittable* for any two-round iterative hash construction.

Lemma 4.3. *The challenge-response game* $\boldsymbol{CRP}_{p,c}^{H^{\mathtt{h}},\mathcal{A}_1,\mathcal{A}_2}$ *(cf. Figure 1) is* $(t_{\mathcal{A}^*}, q_{\mathcal{A}^*}, \epsilon_G, \epsilon_{\mathsf{bad}})$-unsplittable *for any* r*-round iterative hash construction* $H_r^{\mathtt{h}}$ *with* $r \geq 2$*. Moreover,*

$$t_{\mathcal{A}_i^*} \leq t_{\mathcal{A}_i} \qquad q_{\mathcal{A}_i^*} \leq q_{\mathcal{A}_i} \qquad \epsilon_G \leq \frac{q_{\mathcal{A}_1}}{2^{-c}} + \frac{q_{\mathcal{A}_2}}{2^{p-n}} + \frac{q_{\mathcal{A}_1} + q_{\mathcal{A}_2}}{2^s} + \frac{1}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}} \qquad \epsilon_{\mathsf{bad}} = 0$$

Value $t_{\mathcal{A}_i}$ *(resp.* $q_{\mathcal{A}_i}$*) denotes the run-time (resp. number of* $\mathtt{h}$*-queries) of adversary* $\mathcal{A}_i$*.*

Proof. We show that no adversary $(\mathcal{A}_1, \mathcal{A}_2)$ has noticeable probability in winning the CRP game in case the hash functionality is instantiated with a two-round indifferentiable hash construction.

To win in the CRP game with a two-round hash function, $\mathcal{A}_2$ must compute value $H_2^{\mathtt{h}}(M||C)$ which, according to Definition 3.1, can be written as $\mathtt{g}(\mathtt{h}(m_{\mathtt{res}}, x_{\mathtt{res}}))$ where $\mathtt{g}$ is some transformation and (m, x) is the input to the final $\mathtt{h}$-call in the second round of the computation of $H_2^{\mathtt{h}}(M||C)$. By Lemma 3.4 we have that if only a single $\mathtt{h}$-query in the evaluation of $H_2^{\mathtt{h}}(M||C)$ is not queried, then the probability of outputting $H_2^{\mathtt{h}}(M||C)$ is at most

$$\frac{q_{\mathcal{A}_1} + q_{\mathcal{A}_2}}{2^s} + \frac{1}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}} \ .$$

We now argue that adversary $\mathcal{A}_1$ cannot query $\mathtt{h}$-queries in $H_2^{\mathtt{h}}(M||C)$ that correspond $\mathtt{h}$-nodes occurring in the second round. Challenge C is of length c bits and, thus, can be guessed by $\mathcal{A}_1$ with probability at most 2^{-c}. As C will be part of one or multiple message-block-nodes in the execution graph $\mathtt{construct}(M||C)$, we have that $\mathcal{A}_1$ is able to make all $\mathtt{h}$-queries in the first round of $\mathtt{construct}(M||C)$ with probability at most

$$\frac{q_{\mathcal{A}_1}}{2^c} \ .$$

Similarly, adversary $\mathcal{A}_2$ is given only n bits of information from the first round and is, thus, missing at least $p - n$ bits of message M. Thus, the probability, that it is able to make all $\mathtt{h}$-queries in the second round is upper bounded by

$$\frac{q_{\mathcal{A}_2}}{2^{p-n}} \ .$$

If we set adversary $(\mathcal{A}_1^*, \mathcal{A}_2^*)$ as $\mathcal{A}_1^* = \mathcal{A}_1$ and $\mathcal{A}_2^*$ as the procedure that simply outputs a guess for value Z, then we have that $\epsilon_{\mathsf{bad}} = 0$ and

$$\epsilon_G \leq \frac{q_{\mathcal{A}_1}}{2^{-c}} + \frac{q_{\mathcal{A}_2}}{2^{p-n}} + \frac{q_{\mathcal{A}_1} + q_{\mathcal{A}_2}}{2^s} + \frac{1}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}} \ .$$

□

4.2 A Conjecture on Two-Stage Games

In the following we want to present a conjecture on games consisting of exactly two stages. Both the CDA game and the CRP game are examples of a two stage game.

Conjecture 4.4. *Any two-stage functionality-respecting game is* unsplittable *for any* r*-round iterative hash function* $H_r^{\mathtt{h}}$ *with* $r \geq 2$*.*

The idea behind Conjecture 4.4 is simple. In a two-stage game, a bad query can only be made by the second-stage adversary. Let us consider two-round hash functions. Then we can distinguish between two cases in the event that $\mathtt{bad}(m,x)$ occurs for some query (m,x) by $\mathcal{A}_2$. Let $\mathtt{pg}$ be the partial execution graph corresponding to query (m,x). Then, either (m,x) corresponds to an $\mathtt{h}$-node in $\mathtt{pg}$ in the first, or in the second round (the probability of it corresponding to two $\mathtt{h}$-nodes can be upper bounded by the probability of an $\mathtt{h}$-collision).

In the first case, the entire second round is computed by the second stage adversary $\mathcal{A}_2$ and thus it must know the entire message corresponding to the partial graph as all message-block-nodes from round 1 reappear in round 2 (see Definition 3.2). In this case, however, $\mathcal{A}_2$ could have simply computed $H_2^{\mathtt{h}}(M)$ directly. For the second case a similar argument applies. Here the entire first round is computed by $\mathcal{A}_1$ which must hence know the entire message M. As bad queries can only occur with non-negligible probability if there is sufficient communication between the adversaries, adversary $\mathcal{A}_1$ could, thus, have also passed on $H^{\mathtt{h}}(M)$ instead of some intermediate value.

5 Composition for Unsplittable Multi-Stage Games

In this section we present a composition theorem for *unsplittable* games. Intuitively, the composition theorem says that if a game is *unsplittable* for an indifferentiable hash construction $H^{\mathtt{h}}$, then security proofs in the random oracle model carry over if we instantiate the random oracle with hash function $H^{\mathtt{h}}$.

We here give the composition theorem in an asymptotic setting. The full theorem with concrete advantages is given together with its proof in Appendix C (the theorem appears on page 35). We here only present a much shortened proof sketch for the asymptotic version of the theorem.

Theorem 5.1 (Asymptotic Setting). *Let $H^{\mathtt{h}} : \{0,1\}^* \to \{0,1\}^n$ be an iterative hash function indifferentiable from a random oracle $\mathcal{R}$. Let game $G \in \mathcal{LG}$ be any functionality respecting game that is* unsplittable *for $H^{\mathtt{h}}$ and let $\mathcal{A}_1, \ldots, \mathcal{A}_m$ be an adversary. Then, there exists efficient adversary $\mathcal{B}_1, \ldots, \mathcal{B}_m$ and negligible function $\mathtt{negl}$ such that for all values y*

$$\left|\Pr\left[G^{H^{\mathtt{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m} \Rightarrow y\right] - \Pr\left[G^{\mathcal{R},\mathcal{B}_1,\ldots,\mathcal{B}_m} \Rightarrow y\right]\right| \leq \mathtt{negl}(\lambda) \ .$$

Proof Sketch. Since $H^{\mathtt{h}}$ is indifferentiable from a random oracle, there exists an efficient simulator $\mathcal{S}$ and negligible function $\mathtt{negl}$ such that for any efficient distinguisher $\mathcal{D}$

$$\left|\Pr\left[\mathcal{D}^{H^{\mathtt{h}},h}(1^\lambda) = 1\right] - \Pr\left[\mathcal{D}^{\mathcal{R},\mathcal{S}^{\mathcal{R}}}(1^\lambda) = 1\right]\right| \leq \mathtt{negl}(\lambda) \ .$$

In a first step, we construct a simulator $\mathcal{S}_*$ from $\mathcal{S}$. Simulator $\mathcal{S}_*$ answers any repeated query with the same result as the earlier query. For fresh queries it distinguishes between result and non-result queries (for this, it implements a logic similar to the extractor of Lemma 3.5, see Figure 6). It answers any non-result query by a randomly chosen value. For result queries it keeps track of the corresponding partial graph $\mathtt{pg}$ and answers result queries as $\mathtt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathtt{pg})$ where simulator $\mathcal{S}$is the "good" simulator from above. After answering a result query, simulator $\mathcal{S}_*$ resets the state of its internal simulator $\mathcal{S}$.

Result queries will, by construction, be answered correctly (as simulator $\mathcal{S}$ is by definition "good"). Non-result queries, on the other hand, do not help the distinguisher to distinguish the real from the ideal world, that is, a distinguisher here can be reduced to a distinguisher against the underlying simulator $\mathcal{S}$. Let us note that the actual argument is much more involved and we give the full proof and construction in the Appendix as Lemma C.4 and Construction C.3. We have thereby established that there exists a negligible function $\mathtt{negl}$ such that, for any efficient distinguisher $\mathcal{D}$, it holds

$$\left|\Pr\left[\mathcal{D}^{H^{\mathtt{h}},h}(1^\lambda) = 1\right] - \Pr\left[\mathcal{D}^{\mathcal{R},\mathcal{S}_*^{\mathcal{R}}}(1^\lambda) = 1\right]\right| \leq \mathtt{negl}(\lambda) \ .$$

Derandomizing Simulator $\mathcal{S}_*$. In a next step we are going to derandomize simulator $\mathcal{S}_*$. Let $\mathcal{A}_1^*, \ldots, \mathcal{A}_m^*$ be the procedures guaranteed to exist (by Definition 4.1) such that bad queries for hash function $H^{\mathtt{h}}$ occur only with negligible probability during game $G^{H^{\mathtt{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*}$ and that games $G^{H^{\mathtt{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*}$ and $G^{H^{\mathtt{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m}$ are computationally indistinguishable. As adversaries $\mathcal{A}_1^*, \ldots, \mathcal{A}_m^*$ are efficient there exists value $p_{\max}$ (polynomial in the security parameter) denoting the maximum runtime of any of the

adversaries. Simulator $\mathcal{S}_d$ works exactly as simulator $\mathcal{S}_*$ but whenever it wants to return a random value in $\{0,1\}^s$ for some query (m,x), it computes value $r \leftarrow \mathcal{R}(1^{p_{\max}+1})$ and returns value $\mathcal{R}(r||m||x)$. Note that $\mathcal{S}_d$ is still efficient and indistinguishable from simulator $\mathcal{S}_*$ for any distinguisher running in time less than $p_{\max}$ and thus, in particular, for adversaries $\mathcal{A}_1^*, \ldots, \mathcal{A}_m^*$. The derandomization is covered by Lemma C.1 which we believe is of independent interest. Let us not that for result queries the derandomization is a bit more involved as a simulator might have quite some degree of freedom for the choice of its final answer to query $(m_{\mathtt{res}}, x_{\mathtt{res}})$ as it has only to ensure that $\mathsf{g}(m_{\mathtt{res}}, x_{\mathtt{res}}) = \mathcal{R}(M)$ for the corresponding message M. If transformation g, for example, compresses, as is the case in chop-MD, then two different instances of simulator $\mathcal{S}_d$ might give different answers to the same query. We elaborate on this in the full proof in Appendix C and assume for this sketch that result queries are answered independent of the state of the simulator.

We now construct adversaries $\mathcal{B}_1, \ldots, \mathcal{B}_m$ as $\mathcal{B}_i := \mathcal{A}_i^{*\,\mathcal{S}_d^{(i)}}$. That is, adversary $\mathcal{B}_i$ runs adversary $\mathcal{A}_i^*$ with its own instantiation of simulator $\mathcal{S}_d$ answering queries by $\mathcal{S}_d$ to the random oracle with its own access to $\mathcal{R}$. A similar way to perceive this is that we only have a single instance of simulator $\mathcal{S}_d$ which is reset whenever game G calls a new adversarial procedure.

As simulator $\mathcal{S}_d$ is deterministic, different instantiations will produce the same result on the same non-result query. For result queries, we have that different instantiations will produce the same result if all queries in the corresponding execution graph were queried to the simulator instance. Assume that adversarial procedure AdvSeq_i (for $i \in \{1, ..., |\mathsf{AdvSeq}|\}$) makes a result query. Then this query is bad only with negligible probability. The same holds for all queries in the corresponding partial graph $\mathfrak{pg}$ and, thus, all queries have been processed by simulator $\mathcal{S}_d^{(i)}$ in correct order with overwhelming probability and thus all instances of simulator $\mathcal{S}_d$ will answer the result query consistently. If by $\approx$ we denote negligibly close we, thus, have that

$$\begin{aligned}\Pr\left[G^{\mathcal{R},\mathcal{B}_1,\ldots,\mathcal{B}_m} \Rightarrow y\right] \approx \\ \Pr\left[G^{\mathcal{R},\mathcal{A}_1^{*\,\mathcal{S}_d^{(1)}},\ldots,\mathcal{A}_m^{*\,\mathcal{S}_d^{(m)}}} \Rightarrow y\right] \approx \Pr\left[G^{\mathcal{R},\mathcal{A}_1^{*\,\mathcal{S}_d},\ldots,\mathcal{A}_m^{*\,\mathcal{S}_d}} \Rightarrow y\right] \approx \Pr\left[G^{H^{\mathsf{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*} \Rightarrow y\right] \\ \approx \Pr\left[G^{H^{\mathsf{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m} \Rightarrow y\right]\end{aligned}$$

which concludes the proof. ◊

Acknowledgments

I thank the anonymous reviewers for valuable comments. This work was supported by CASED (`www.cased.de`).

References

[1] Andreeva, E., Bogdanov, A., Mennink, B., Preneel, B., Rechberger, C.: On security arguments of the second round SHA-3 candidates. Cryptology ePrint Archive, Report 2012/147 (2012), `http://eprint.iacr.org/` (Cited on page 24.)

[2] Andreeva, E., Mennink, B., Preneel, B.: On the indifferentiability of the Grøstl hash function. In: Garay, J.A., Prisco, R.D. (eds.) SCN 10: 7th International Conference on Security in Communication Networks. Lecture Notes in Computer Science, vol. 6280, pp. 88–105. Springer, Berlin, Germany, Amalfi, Italy (Sep 13–15, 2010) (Cited on page 3.)

[3] Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: SHA-3 proposal BLAKE. Submission to NIST (Round 3) (2010), `http://131002.net/blake/blake.pdf` (Cited on page 5.)

[4] Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: Boyd, C. (ed.) Advances in Cryptology – ASIACRYPT 2001. Lecture Notes in Computer Science, vol. 2248, pp. 566–582. Springer, Berlin, Germany, Gold Coast, Australia (Dec 9–13, 2001) (Cited on pages 4, 17, 45, and 46.)

[5] Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and efficiently searchable encryption. In: Menezes, A. (ed.) Advances in Cryptology – CRYPTO 2007. Lecture Notes in Computer Science, vol. 4622, pp. 535–552. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 19–23, 2007) (Cited on pages 3, 4, 38, 39, and 41.)

[6] Bellare, M., Brakerski, Z., Naor, M., Ristenpart, T., Segev, G., Shacham, H., Yilek, S.: Hedged public-key encryption: How to protect against bad randomness. In: Matsui, M. (ed.) Advances in Cryptology – ASIACRYPT 2009. Lecture Notes in Computer Science, vol. 5912, pp. 232–249. Springer, Berlin, Germany, Tokyo, Japan (Dec 6–10, 2009) (Cited on pages 3, 4, 16, 38, 41, 44, 45, and 47.)

[7] Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) Advances in Cryptology – CRYPTO'96. Lecture Notes in Computer Science, vol. 1109, pp. 1–15. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 18–22, 1996) (Cited on pages 4, 6, and 7.)

[8] Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Ashby, V. (ed.) ACM CCS 93: 1st Conference on Computer and Communications Security. pp. 62–73. ACM Press, Fairfax, Virginia, USA (Nov 3–5, 1993) (Cited on page 3.)

[9] Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) Advances in Cryptology – EUROCRYPT 2006. Lecture Notes in Computer Science, vol. 4004, pp. 409–426. Springer, Berlin, Germany, St. Petersburg, Russia (May 28 – Jun 1, 2006) (Cited on pages 6 and 25.)

[10] Bennett, C.H., Gill, J.: Relative to a random oracle A, $P^A \neq NP^A \neq coNP^A$ with probability 1. SIAM Journal on Computing 10(1), 96–113 (1981) (Cited on page 27.)

[11] Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The keccak SHA-3 submission. Submission to NIST (Round 3) (2011), `http://keccak.noekeon.org/Keccak-submission-3.pdf` (Cited on pages 3 and 5.)

[12] Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Cryptographic sponge functions (2011) (Cited on pages 3, 5, and 24.)

[13] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the indifferentiability of the sponge construction. In: Smart, N.P. (ed.) Advances in Cryptology – EUROCRYPT 2008. Lecture Notes in Computer Science, vol. 4965, pp. 181–197. Springer, Berlin, Germany, Istanbul, Turkey (Apr 13–17, 2008) (Cited on page 3.)

[14] Bhattacharyya, R., Mandal, A., Nandi, M.: Indifferentiability characterization of hash functions and optimal bounds of popular domain extensions. In: Roy, B.K., Sendrier, N. (eds.) Progress in Cryptology - INDOCRYPT 2009: 10th International Conference in Cryptology in India. Lecture Notes in Computer Science, vol. 5922, pp. 199–218. Springer, Berlin, Germany, New Delhi, India (Dec 13–16, 2009) (Cited on page 3.)

[15] Chang, D., Nandi, M., Yung, M.: Indifferentiability of the hash algorithm BLAKE. Cryptology ePrint Archive, Report 2011/623 (2011), http://eprint.iacr.org/ (Cited on page 3.)

[16] Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård revisited: How to construct a hash function. In: Shoup, V. (ed.) Advances in Cryptology – CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621, pp. 430–448. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 14–18, 2005) (Cited on pages 3, 5, 6, and 30.)

[17] Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) Advances in Cryptology – CRYPTO'98. Lecture Notes in Computer Science, vol. 1462, pp. 13–25. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 23–27, 1998) (Cited on page 45.)

[18] Damgård, I.: A design principle for hash functions. In: Brassard, G. (ed.) Advances in Cryptology – CRYPTO'89. Lecture Notes in Computer Science, vol. 435, pp. 416–427. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 20–24, 1990) (Cited on pages 3, 5, and 7.)

[19] Demay, G., Gaži, P., Hirt, M., Maurer, U.: Resource-restricted indifferentiability. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. Lecture Notes in Computer Science, vol. 7881, pp. 665–684. Springer, Berlin, Germany, Athens, Greece (May 26–30, 2013) (Cited on pages 4 and 6.)

[20] Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The skein hash function family. Submission to NIST (Round 3) (2010), http://www.skein-hash.info/sites/default/files/skein1.3.pdf (Cited on page 5.)

[21] Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schlffer, M., Thomsen, S.S.: Grstl – a SHA-3 candidate. Submission to NIST (Round 3) (2011), http://www.groestl.info/Groestl.pdf (Cited on page 5.)

[22] Joux, A.: Multicollisions in iterated hash functions. application to cascaded constructions. In: Franklin, M. (ed.) Advances in Cryptology – CRYPTO 2004. Lecture Notes in Computer Science, vol. 3152, pp. 306–316. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 15–19, 2004) (Cited on page 25.)

[23] Liskov, M.: Constructing an ideal hash function from weak ideal compression functions. In: Biham, E., Youssef, A.M. (eds.) SAC 2006: 13th Annual International Workshop on Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 4356, pp. 358–375. Springer, Berlin, Germany, Montreal, Canada (Aug 17–18, 2006) (Cited on pages 3, 5, and 9.)

[24] Lucks, S.: Design principles for iterated hash functions. Cryptology ePrint Archive, Report 2004/253 (2004), http://eprint.iacr.org/ (Cited on pages 24, 25, and 27.)

[25] Luykx, A., Andreeva, E., Mennink, B., Preneel, B.: Impossibility results for indifferentiability with resets. Cryptology ePrint Archive, Report 2012/644 (2012), http://eprint.iacr.org/ (Cited on pages 4 and 6.)

[26] Maurer, U.M., Renner, R., Holenstein, C.: Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In: Naor, M. (ed.) TCC 2004: 1st Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 2951, pp. 21–39. Springer, Berlin, Germany, Cambridge, MA, USA (Feb 19–21, 2004) (Cited on pages 3, 5, and 6.)

[27] Merkle, R.C.: One way hash functions and DES. In: Brassard, G. (ed.) Advances in Cryptology – CRYPTO'89. Lecture Notes in Computer Science, vol. 435, pp. 428–446. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 20–24, 1990) (Cited on pages 3, 5, and 7.)

[28] Moody, D., Paul, S., Smith-Tone, D.: Improved indifferentiability security bound for the JH mode. Cryptology ePrint Archive, Report 2012/278 (2012), http://eprint.iacr.org/ (Cited on page 3.)

[29] NIST: NIST SHA-3 Competition, http://csrc.nist.gov/groups/ST/hash/sha-3/index.html (Cited on page 3.)

[30] Ristenpart, T., Shacham, H., Shrimpton, T.: Careful with composition: Limitations of indifferentiability and universal composability. Cryptology ePrint Archive, Report 2011/339 (2011), http://eprint.iacr.org/ (Cited on pages 6 and 37.)

[31] Ristenpart, T., Shacham, H., Shrimpton, T.: Careful with composition: Limitations of the indifferentiability framework. In: Paterson, K.G. (ed.) Advances in Cryptology – EUROCRYPT 2011. Lecture Notes in Computer Science, vol. 6632, pp. 487–506. Springer, Berlin, Germany, Tallinn, Estonia (May 15–19, 2011) (Cited on pages 2, 3, 4, 6, 14, 16, 25, 27, 37, 38, 40, 41, and 42.)

[32] Rivest, R.: The MD5 Message-Digest Algorithm. RFC 1321 (Informational) (Apr 1992), http://www.ietf.org/rfc/rfc1321.txt, updated by RFC 6151 (Cited on pages 3 and 5.)

[33] Winternitz, R.S.: Producing a one-way hash function from DES. In: Chaum, D. (ed.) Advances in Cryptology – CRYPTO'83. pp. 203–207. Plenum Press, New York, USA, Santa Barbara, CA, USA (1984) (Cited on page 5.)

[34] Wu, H.: The hash function JH. Submission to NIST (round 3) (2011), http://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf (Cited on page 5.)

A Formalizing Iterative Hash Functions

A.1 Execution Graphs

In the following section we formally describe execution graphs for iterative hash functions (see Section 3). We describe the structure of the execution graph for message $m_1 \| \ldots \| m_\ell := \mathtt{pad}(M)$. An execution graph is a directed graph where nodes represent constants or functions while edges define the evaluation path. An execution graph contains exactly one unbound outgoing edge. The graph consists of the following node and edge types:

$\mathcal{IV}$-node: For every string $\mathsf{iv} \in \mathcal{IV}$ there can exist an $\mathcal{IV}$-node with in-degree 0. Outgoing edges are of type h-edge or m-edge labeled with value iv. If the outgoing edge is of type h-edge (resp. m-edge) it must hold that $\mathsf{iv} \in \{0,1\}^s$ (resp. $\mathsf{iv} \in \{0,1\}^b$).

$\mathcal{C}$-node: For every string $c \in \mathcal{C}$ there can exist a $\mathcal{C}$-node with in-degree 0. Outgoing edges are of type h-edge or m-edge labeled with value c. If the outgoing edge is of type h-edge (resp. m-edge) it must hold that $c \in \{0,1\}^s$ (resp. $c \in \{0,1\}^b$).

message-node: For every message block m_i (for $1 \leq i \leq \ell$) there exists a node with in-degree 0 and out-degree at least 1. Outgoing edges are of type m-edge labeled with value $\langle i \rangle_{\lceil \log \ell \rceil} \| m_i$.

mp-node: A mp-node has in-degree 1 which takes an m-edge and out-degree 1. The outgoing edge is of type mp-edge.

hp-node: A hp-node has in-degree 1 taking an h-edge and out-degree 1. The outgoing edge is of type hp-edge.

hmp-node: A hmp-node has in-degree 1 taking an h-edge and out-degree 1. The outgoing edge is of type mp-edge. We add the additional restriction that ingoing h-edges may not come from $\mathcal{IV}$-nodes or $\mathcal{C}$-nodes.

h-nodes: An h-node has in-degree 2, a mp-edge and a hp-edge, and has out-degree at least 1. Outgoing edges are of type h-edge. We add the additional restriction that if the corresponding hp-node is connected to a $\mathcal{C}$-node then the second ingoing edge of type mp must be connected to an hmp-node. Additionally, if the if the corresponding hp-node is connected to an $\mathcal{IV}$-node then the second ingoing edge of type mp must be connected to a mp-node.

g-node: There exists a single g-node with in-degree 1, taking an h-edge and out-degree 1. The outgoing edge is not connected to a node.

We call $\mathcal{IV}, \mathcal{C}$, and message-nodes *value-nodes* and all other node types *function-nodes*. All outgoing edges must be connected to a node with the only exception being the outbound edge from the single g-node. We say that an execution graph uses *single-$\mathcal{IV}$-mode* if all $\mathcal{IV}$-nodes are connected to hp-nodes. We say the execution graph is in *full-$\mathcal{IV}$-mode*, if it holds for all h-nodes that are connected to an $\mathcal{IV}$-node via a hp-node that also the second ingoing mp-edge comes from a mp-node which in turn is connected to an $\mathcal{IV}$-node.

A *valid execution graph* is a graph that is not empty and complies with the above rules. For each message $M \in \{0,1\}^*$ there is exactly one valid execution graph. We will also need the concept of *partial execution graphs* which is a non-empty graph that complies to the above specified rules with the only exception that it does not contain a g-node. However, it must contain exactly one unbound outgoing h-edge.

We define EVAL to be a generic, deterministic algorithm evaluating execution graphs relative to an oracle h. Let pg be a an execution graph for some message $M \in \{0,1\}^*$. To evaluate pg relative to oracle h, algorithm $\mathtt{EVAL}^{\mathtt{h}}(\mathtt{pg})$ first verifies the graph structure validating, that pg obeys the rules as specified by the hash construction and is either a valid execution graph or a partial execution graph. It then performs the following steps to compute the hash value: search for a node that has no inbound edges or for which all inbound edges are labeled. If the node is a value node, then remove the node (in this case the outgoing edges are already labeled). If the node is a function node then evaluate the corresponding function using the labels from the inbound edges as input. Remove the node from the graph and label

Algorithm: $\texttt{EVAL}^h(\texttt{pg})$

$y \leftarrow \perp$
if (pg is not correct partial graph) **then return** y
while (pg contains nodes) **do**
  foreach (node in pg) **do**
    if (node is value-node) **then** remove node from pg
    if (node is function-node $\wedge$ all inbound edges are labeled) **then**
      $y \leftarrow \mathsf{evaluate}^{\texttt{h}}(\texttt{node})$
      label all outgoing edges of node with y
      remove node and inbound edges from pg
return y

Algorithm: $\mathsf{evaluate}^{\mathcal{O}}(\texttt{node})$

if (node is mp-node) **then**
  return mp(node.in-m)
if (node is hp-node) **then**
  return hp(node.in-h)
if (node is hmp-node) **then**
  return hmp(node.in-h)
if (node is h-node) **then**
  $m \leftarrow$ node.in-mp
  $x \leftarrow$ node.in-hp
  return $\mathcal{O}(m, x)$
if (node is g-node) **then**
  return g(node.in-h)

Figure 7: The generic evaluation algorithm $\texttt{EVAL}^h$. For the evaluation of function nodes we denote by node.in-T the label of the ingoing edge of type T.

all outgoing edges with the result. If the last node in the graph was removed stop and return the result. Note that $\texttt{EVAL}^{\texttt{h}}(\texttt{pg})$ runs in time at most $\mathcal{O}\left(|V^2|\right)$ assuming that pg contains $|V|$ many nodes. Note that if pg is a partial execution graph then $\texttt{EVAL}^{\texttt{h}}(\texttt{pg})$, likewise, computes the partial graph outputting the result of the final h-node. Further, if pg is a partial execution graph, then we denote by $\texttt{g}(\texttt{pg})$ the corresponding execution graph where the single outbound h-edge of pg is connected to a g-node. We call this the *completed* execution graph for pg. We give the pseudo-code of algorithm EVAL in Figure 7.

Extensions to the model. In the above model, we have defined the preprocessing nodes to have in-degree 1. For certain constructions (such as, for example, the double-pipe construction [24], see Section A.2.4) this requirement needs to be relaxed. Such relaxations slightly complicate the definition of initial, chained and result queries (see Section 3.1) but do not change the presented results (in an asymptotic setting).

A.2 Examples: Hash Constructions in Compliance with Definition 3.1

A.2.1 Merkle-Damgård-like Functions

In the following we show that Merkle-Damgård-like functions such as the plain or chop-MD constructions, but also the Sponge construction [12] used in SHA-3 are covered by Definition 3.1. The difference between chop-MD and the plain Merkle-Damgård construction only lies in the final transformation g which is the identity for plain Merkle-Damgård and which truncates the output of the final compression function call in case of chop-MD. Note that the Sponge construction can be regarded as chop-MD if we restrict ourselves to a single "squeezing" round [1].

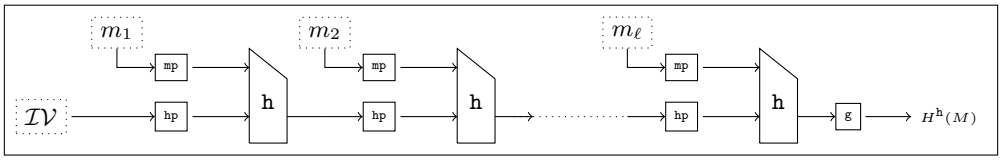

Figure 8: Merkle-Damgård Construction

Merkle-Damgård constructions use single-$\mathcal{IV}$-mode, that is, the $\mathcal{IV}$ is only connected to a hp-node and the corresponding mp-node takes a message block. Given message blocks $m_1 \| \ldots \| m_\ell = \texttt{pad}(M)$ for $M \in \{0,1\}^*$ it is easy to see that we can construct the corresponding execution graph in time linear in the number of message blocks, that is $\mathcal{O}\left(\frac{|M|}{d}\right)$. Extracting the message from a graph, as well as verifying that a graph is a correct partial execution graph can be done in time linear in the number of nodes.

A.2.2 NMAC and HMAC

In the following we show how NMAC and HMAC fit into Definition 3.1. Note that this is the first construction where we use the `hmp` preprocessing function as well as constants from set $\mathcal{C}$. The running times of `construct` and `extract` are equivalent to the running times for the basic Merkle-Damgård constructions. While NMAC again uses single-$\mathcal{IV}$-mode, HMAC uses full-$\mathcal{IV}$-mode, where in the $\mathcal{IV}$-stage both inputs to the `h`-node are connected to $\mathcal{IV}$-nodes.

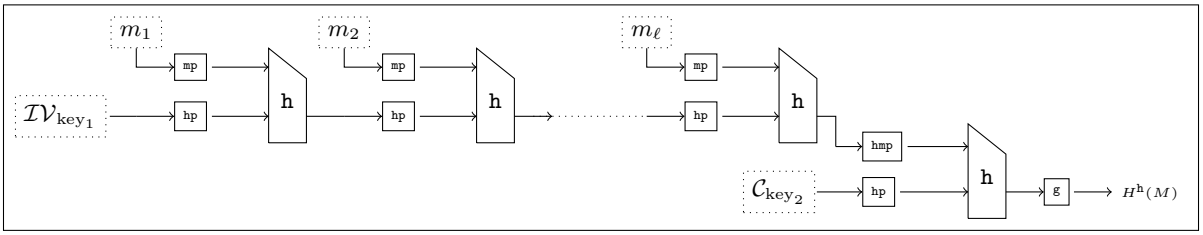

Figure 9: NMAC: note the use of the `hmp` node to connect the final `h`-node to the plain Merkle-Damgård construction. Also note the use of the $\mathcal{C}$-node instead of an $\mathcal{IV}$-node for the second key. This is necessary such that the final `h`-query is not an initial query, which given our definitions would be the case if key_2 was part of $\mathcal{IV}$.

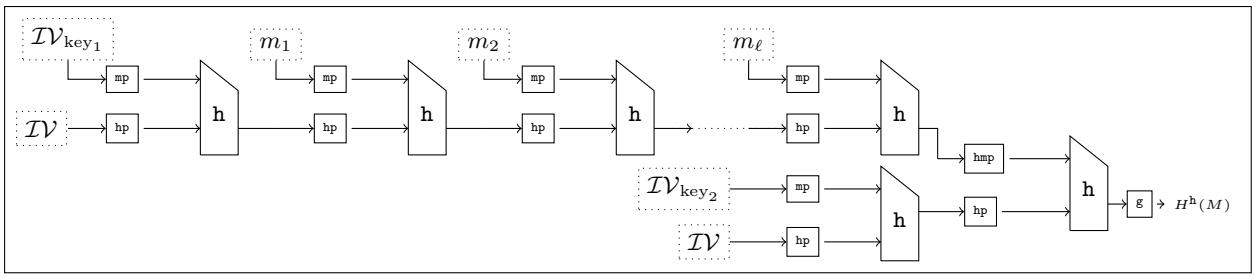

Figure 10: The HMAC construction. Note that HMAC is similar to NMAC except that keys key_1 and key_2 are no generated by `h`-calls.

A.2.3 Hash Tree

In Figure 11 we show how hash trees fit into Definition 3.1. For simplicity we assume that the number of message blocks ℓ is a power of 2. Given message blocks $m_1 \| \ldots \| m_\ell = \texttt{pad}(M)$ for $M \in \{0,1\}^*$ it is easy to see that we can construct the corresponding execution graph in time log-linear in the number of message blocks, that is $\mathcal{O}(\ell \log \ell)$. Extracting the message of a given partial graph can be done in time of a (reverse) breadth first search starting from the final `h`-node.

A.2.4 The Double-Pipe Construction / Extensions to the Model

Stefan Lucks [24] proposes several tweaks to the design of iterated hash functions to, for example, rule out generic attacks such as Joux' multi-collision attack [22]. In Figure 12 we show how the double-pipe construction fits into Definition 3.1. Note that to support constructions like the double-pipe construction we must slightly extend our model of iterative hash functions. We must now allow that `hp`-nodes not only have in-degree 1 but 2. Besides slightly complicating the definition of initial queries and chained queries —we now have to split the pre-image of $\texttt{hp}^{-1}(x)$ into multiple values— the proofs and intuition presented in this paper work analogously.

B Game Playing

For the discussion in this paper we use the game playing technique as described in [9, 31]. Games consist of procedures which in turn consist of a sequence of statements together with some input and zero or more outputs. Procedures can call other procedures. If procedures P_1 and P_2 have inputs and outputs

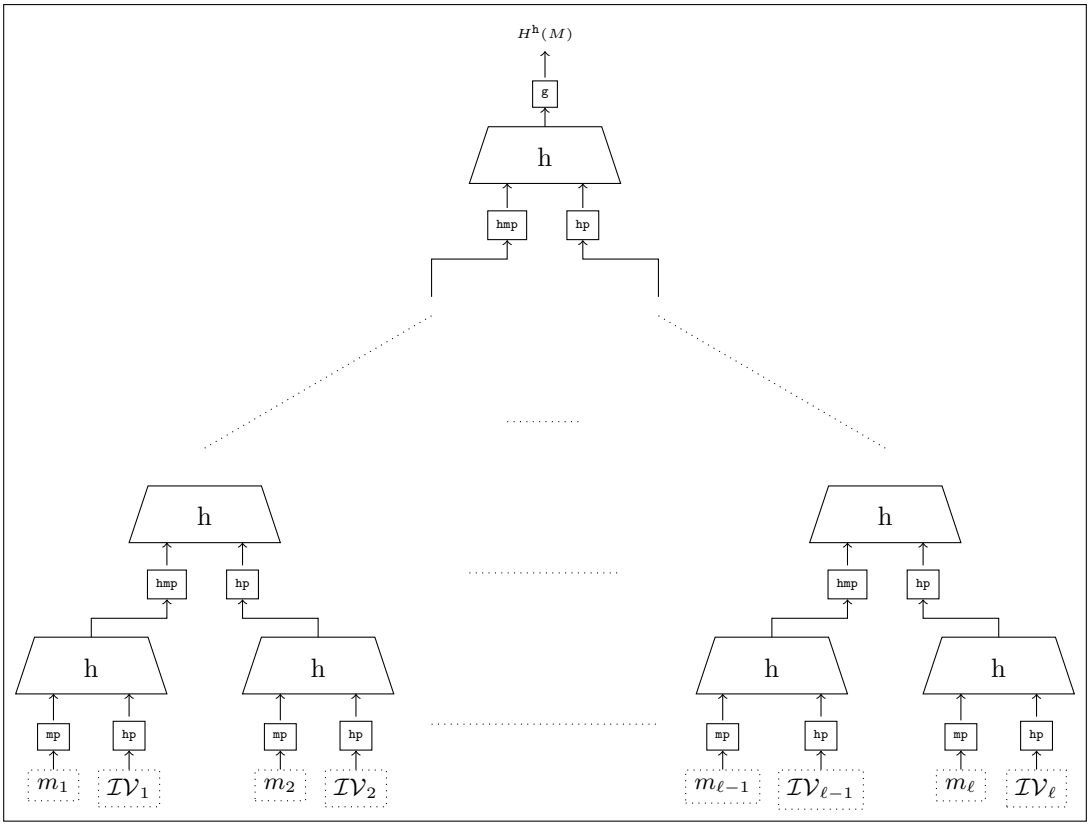

Figure 11: A hash tree in the formalization of Definition 3.1.

that are identical in number and type, we say that they export the same interface. If a procedure P gets access to procedure $\mathcal{F}$ we denote this by adding it in superscript $P^{\mathcal{F}}$. All variables used by procedures are assumed to be of local scope. After the execution of a procedure the variable values are left as they were after the execution of the last statement. If procedures are called multiple times, this allows them to keep track of their state.

A functionality $\mathcal{F}$ is a collection of two procedures $\mathcal{F}$.hon and $\mathcal{F}$.adv, with suggestive names "honest" and "adversarial". Adversaries access a functionality $\mathcal{F}$ via the interface exported by $\mathcal{F}$.adv, while all other procedures access the functionality via $\mathcal{F}$.hon. In our case, functionalities are hash functionalities which will either be instantiated with typical iterative hash constructions or with random oracles. For iterative hash constructions the adversarial interface accesses the compression function and the honest interface provides access to the complete hash function as specified, i.e., $\mathcal{F}.hon := H^{\mathtt{h}}$ and $\mathcal{F}.adv := h$. Note that access to the compression function is sufficient to compute $H^{\mathtt{h}}$. For a random oracle, on the other hand, there is no distinction between adversary access and honest access and we can assume that the adversarial interface simply forwards calls to the honest interface. As in this paper we are solely talking about iterative hash functions we will usually not write $\mathcal{F}$.hon and $\mathcal{F}$.adv, but directly refer to hash function $H^{\mathtt{h}}$ and underlying function $\mathtt{h}$, respectively.

A game G consists of a distinguished procedure called **main** (which takes no input) together with a set of procedures. A game can make use of functionality $\mathcal{F}$ and adversarial procedures $\mathcal{A}_1, \ldots, \mathcal{A}_m$ (together called "the adversary"). Adversarial procedures have access to the adversarial interface of functional procedures and, as any other procedure, can be called multiple times. We, however, restrict access to adversarial procedures to the game's **main** procedure, i.e., only it can call adversarial procedures and, in particular, adversarial procedures cannot call one another directly.

By $G^{\mathcal{F},\mathcal{A}_1,\ldots,\mathcal{A}_m}$ we denote a game using functionality $\mathcal{F}$ and adversary $\mathcal{A}_1, \ldots, \mathcal{A}_m$. If $\mathcal{F}'$ exports the same interface as $\mathcal{F}$, and for $1 \leq i \leq m$ adversary $\mathcal{A}'_i$ exports the same interface as $\mathcal{A}_i$, then $G^{\mathcal{F}',\mathcal{A}'_1,\ldots,\mathcal{A}'_m}$ executes the same game G with functional procedure $\mathcal{F}'$ and adversary $\mathcal{A}'_1, \ldots, \mathcal{A}'_m$. We denote by $G^{\mathcal{F},\mathcal{A}_1,\ldots,\mathcal{A}_m} \Rightarrow y$ the event that game G produces output y, that is procedure **main** returns value y. If game G uses any probabilistic procedure then $G^{\mathcal{F},\mathcal{A}_1,\ldots,\mathcal{A}_m}$ is a random variable and by

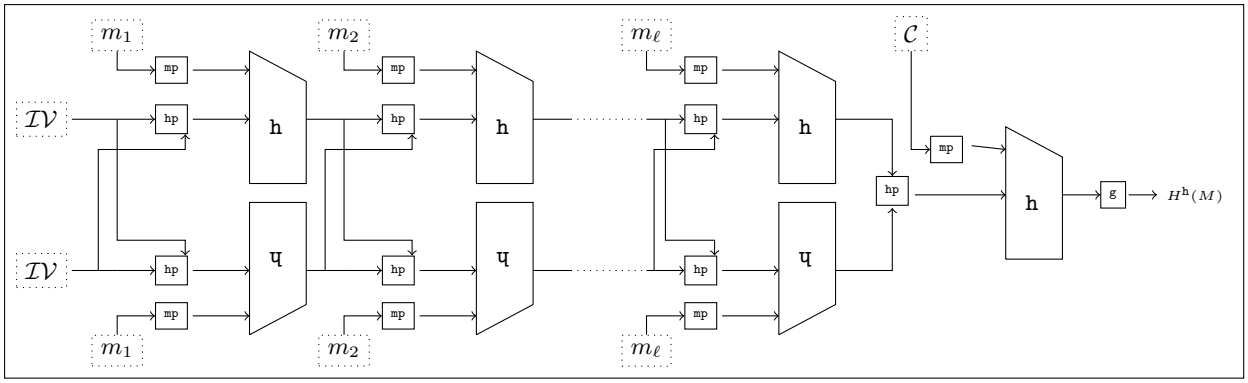

Figure 12: The double-pipe construction from [24]. Note that the message-block-nodes are split for better drawing. They should, however, be regarded as a single node.

$\Pr\left[G^{\mathcal{F},\mathcal{A}_1,\ldots,\mathcal{A}_m} \Rightarrow y\right]$ we denote the probability (over the combined randomness space of the game) that it takes on value y.

Games are random variables over the entire random coins of the game and the adversarial procedures. For functionalities $\mathcal{F}$ and $\mathcal{F}'$ and adversaries $\mathcal{A}_1,\ldots,\mathcal{A}_m$ and $\mathcal{A}'_1,\ldots,\mathcal{A}'_m$, we can thus consider the distance between the two random variables. Our security approach is that of concrete security, i.e., we say two games are ϵ-close if for all values y it holds that

$$\Pr\left[G^{\mathcal{F},\mathcal{A}_1,\ldots,\mathcal{A}_m} \Rightarrow y\right] \leq \Pr\left[G^{\mathcal{F}',\mathcal{A}'_1,\ldots,\mathcal{A}'_m} \Rightarrow y\right] + \epsilon \ .$$

In asymptotic terms this means that if ϵ is negligible in the security parameter, then it follows that for all efficient distinguishers the two games are indistinguishable:

$$\left|\Pr\left[\mathcal{D}(G^{\mathcal{F},\mathcal{A}_1,\ldots,\mathcal{A}_m}, 1^\lambda) = 1\right] - \Pr\left[\mathcal{D}(G^{\mathcal{F}',\mathcal{A}'_1,\ldots,\mathcal{A}'_m}, 1^\lambda) = 1\right]\right| \leq \epsilon(\lambda) \ .$$

Functionality Respecting Games. In this paper we only consider the class of functionality-respecting games $\mathcal{LG}$ as defined by Ristenpart et al. [31]. A game is called *functionality respecting* if only adversarial procedures can call the adversarial interface of functionalities. Note that this restriction is quite natural if a game is used to specify a security goal in the random oracle model since random oracles do not provide any adversarial interface.

C The Composition Theorem 5.1

C.1 Derandomizing Simulators

Before we can prove Theorem 5.1 we need some additional characterization on strong indifferentiability for hash constructions which we believe to be of independent interest. With the following lemma we establish that if $H^{\mathtt{h}}$ is a indifferentiable hash construction then we can derandomize the corresponding simulator. For this we build on derandomization techniques developed by Bennet and Gill [10].

Lemma C.1. *Let $H^{\mathtt{h}}$ be an iterative hash construction and $\mathcal{R}$ a random oracle. Let $\mathcal{S}$ be a simulator exporting the same interface as* h*. Fix $t_{\mathcal{D}} \in \mathbb{N}$. Then there exists* deterministic *simulator $\mathcal{S}_d$ such that for any distinguisher $\mathcal{D}$ with run-time bounded by $t_{\mathcal{D}}$ it holds that*

$$\Pr\left[\mathcal{D}^{\mathcal{R},\mathcal{S}^{\mathcal{R}}_d}(1^\lambda) = 1\right] = \Pr\left[\mathcal{D}^{\mathcal{R},\mathcal{S}^{\mathcal{R}}}(1^\lambda) = 1\right] \ .$$

Simulator $\mathcal{S}_d$ runs in time at most $t_{\mathcal{S}} + t_{\mathcal{S}}(t_{\mathcal{D}} + t_{\mathcal{S}})$ and makes at most $q_{\mathcal{S}} + t_{\mathcal{S}}$ many queries to the random oracle.

Proof. We construct deterministic simulator $\mathcal{S}_d$ as follows. Simulator $\mathcal{S}_d$ works exactly as simulator $\mathcal{S}$ except that when $\mathcal{S}$ requests a random bit, simulator $\mathcal{S}_d$ generates this bit deterministically using its

access to the random oracle. For this it generates a bit stream as

$$\mathcal{R}(1^{t_{\mathcal{D}}+1})\|\mathcal{R}(1^{t_{\mathcal{D}}+2})\|\ldots$$

picking the i-th bit of the stream as the i-th random bit. If we denote with R the random variable, mapping to the random bits used by simulator $\mathcal{S}$ and by $R_d^{\mathcal{R}}$ the random variable, mapping to the coins used by deterministic simulator $\mathcal{S}_d$ (over the choice of random oracle) then their statistical distance (denoted by $\delta(\cdot,\cdot)$) is zero, that is:

$$\delta(R, R_d^{\mathcal{R}}) := \frac{1}{2}\sum_x \left|\Pr[\,R = x\,] - \Pr_{\mathcal{R}}\left[\,R_d^{\mathcal{R}} = x\,\right]\right| = 0$$

As furthermore the queries to generate the random bits are larger than any queries made by any distinguisher with run time bounded by $t_{\mathcal{D}}$ it follows that

$$\Pr\left[\,\mathcal{D}^{\mathcal{R},\mathcal{S}_d^{\mathcal{R}}}(1^\lambda) = 1\,\right] = \Pr\left[\,\mathcal{D}^{\mathcal{R},\mathcal{S}^{\mathcal{R}}}(1^\lambda) = 1\,\right]$$

Assuming each call to the random oracle yields only a single random bit for the bit-stream we have that $\mathcal{S}_d$ has a run-time of $t_{\mathcal{S}} + t_{\mathcal{S}}(t_{\mathcal{D}} + t_{\mathcal{S}})$. It runs the same program as the original simulator taking time at most $t_{\mathcal{S}}$ and has to gather at most $t_{\mathcal{S}}$ many random bits each needing a single call to the random oracle on a string of length at most $t_{\mathcal{D}} + t_{\mathcal{S}}$. □

Note that instead of fixing the runtime of the simulator as $t_{\mathcal{D}}$ and querying the random oracle on very long messages— that is $\mathcal{R}(1^{t_{\mathcal{D}}+1})$ —we could also fix a bound $q_{\mathcal{D}}$ on the number of random oracle queries by the distinguisher. The simulator then makes $q_{\mathcal{D}} + 1$ many distinct messages taking the exclusive or over all results. For any distinguisher that makes at most q many random oracle queries the xor of $q_{\mathcal{D}}+1$ will be uniformly distributed and completely unpredictable. This allows a trade-off between run-time and the number of queries. We give the statement in the following lemma:

Lemma C.2. *Let $H^{\mathtt{h}}$ be an iterative hash construction and $\mathcal{R}$ a random oracle. Let $\mathcal{S}$ be a simulator exporting the same interface as* h. *Fix $q_{\mathcal{D}} \in \mathbb{N}$. Then there exists* deterministic *simulator $\mathcal{S}_d$ such that for any distinguisher $\mathcal{D}$ which makes at most $q_{\mathcal{D}}$ queries to the random oracle, it holds that*

$$\Pr\left[\,\mathcal{D}^{\mathcal{R},\mathcal{S}_d^{\mathcal{R}}}(1^\lambda) = 1\,\right] = \Pr\left[\,\mathcal{D}^{\mathcal{R},\mathcal{S}^{\mathcal{R}}}(1^\lambda) = 1\,\right] .$$

Simulator $\mathcal{S}_d$ runs in time $\mathcal{O}(t_{\mathcal{S}} + q_{\mathcal{D}}\log(q_{\mathcal{D}}))$.

C.2 A Generic Indifferentiabilitiy Simulator

In the following we show that for any indifferentiable hash construction we can use a generic simulator which replies with randomly chosen values on any non result query and which uses the underlying simulator (guaranteed to exist by the fact that the hash function is indifferentiable) to answer result queries. For this we consider the following simulator $\mathcal{S}_*$ that is built from some underlying simulator $\mathcal{S}$. Simulator $\mathcal{S}^*$ will be somewhat similar to the extractor of Lemma 3.5 (see Figure 6). We also use a similar syntax as for the proof of Lemma 3.5.

Construction C.3. *Simulator $\mathcal{S}_*$ is build from simulator $\mathcal{S}$. We give the pseudo-code of simulator $\mathcal{S}_*$ when receiving query (m, x) in Figure 13. We assume that it initializes table $\mathcal{M} \leftarrow []$ and set $\mathcal{PG} \leftarrow \{\}$ before processing the first query.*

As in Lemma 3.5 we denote the value of the sole outgoing h-edge of a partial graph pg, relative to an execution with simulator $\mathcal{S}_*$, by pg.y (which is assigned in line 16). This value is used, as in extractor $\mathcal{E}$ (see Lemma 3.5) to check whether partial graphs can be extended (lines 6 and 9).

Simulator $\mathcal{S}_*$ keeps a table $\mathcal{M}$ for storing queries and a set $\mathcal{PG}$ for storing partial graphs. On receiving a query (m, x) simulator $\mathcal{S}_*$ checks table $\mathcal{M}$ whether the query has been queried before. If so it returns the same result as before: $\mathcal{M}[m, x]$. If it is a new query it takes similar steps as extractor $\mathcal{E}$ in Lemma 3.5 to generate a new partial graph. Note that given the code of the simulator it is possible that more than one new partial graph is generated during the processing of a query (see the foreach loops in lines 6 and

Simulator $\mathcal{S}_*(m,x)$:

1 **if** $\mathcal{M}[m,x] \neq \bot$ **then return** $\mathcal{M}[m,x]$
2 $\mathfrak{newG} \leftarrow \bot$
3 **if** $\mathsf{init}(m,x)$ **then**
4 $\quad \mathfrak{newG} \leftarrow$ **new** PartialGraph$(\mathtt{mp}^{-1}(m), \mathtt{hp}^{-1}(x))$
5 **else**
6 $\quad$ **foreach** $(\mathfrak{pg}, \mathfrak{pg}') \in \mathcal{PG} \times \mathcal{PG} : \mathfrak{pg}.\mathsf{y} = \mathtt{hmp}^{-1}(m) \wedge \mathfrak{pg}'.\mathsf{y} = \mathtt{hp}^{-1}(x)$ **do**
7 $\quad\quad \mathfrak{newG} \leftarrow \mathfrak{pg}.\mathrm{extendedBy}(\mathfrak{pg}', m, x, y))$
8 $\quad$ **if** $\mathfrak{newG} = \bot$ **then**
9 $\quad\quad$ **foreach** $\mathfrak{pg} \in \mathcal{PG} : \mathfrak{pg}.\mathsf{y} = \mathtt{hp}^{-1}(x)$ **do**
10 $\quad\quad\quad \mathfrak{newG} \leftarrow \mathfrak{pg}.\mathrm{extendedBy}(m, x, y)$
11 **if** $\mathfrak{newG} \neq \bot \wedge \mathtt{extract}(\mathfrak{newG}) \neq \bot$ **then**
12 $\quad \mathcal{M}[m,x] \leftarrow \mathtt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})$
13 **else**
14 $\quad \mathcal{M}[m,x] \leftarrow \{0,1\}^s$
15 **if** $\mathfrak{newG} \neq \bot$ **then**
16 $\quad \mathfrak{newG}.\mathsf{y} \leftarrow \mathcal{M}[m,x]$
17 $\quad \mathcal{PG}.\mathrm{add}(\mathfrak{newG})$
18 reset simulator $\mathcal{S}$
19 **return** $\mathcal{M}[m,x]$;

Figure 13: Simulator $\mathcal{S}_*$ from Construction C.3 as pseudo-code.

9). We will later show that this only happens with small probability. If a new partial graph was generated and $\mathtt{extract}(\mathfrak{pg}) \neq \bot$, that is the graph can be completed (see description of model in Section 3), it evaluates the partial graph using $\mathcal{M}(m,x) \leftarrow \mathtt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})$ giving the evaluation algorithm oracle access to the underlying simulator $\mathcal{S}^{\mathcal{R}}$. Note that this execution branch corresponds to result queries. If no partial graph was generated, then $\mathcal{S}_*$ chooses a value $\mathcal{M}(m,x) \leftarrow \{0,1\}^s$ uniformly at random. Finally, if a partial graph $\mathfrak{newG}$ was generated (independent of wether it can be completed or not) it sets value $\mathfrak{newG}.\mathsf{y}$ to $\mathcal{M}(m,x)$ and adds the newly created partial graph to its set of graphs $\mathcal{PG}$ (lines 16 and 17). It then resets the underlying simulator $\mathcal{S}$ and returns value $\mathcal{M}[m,x]$.

In the following we show, that simulator $\mathcal{S}_*$ as described above is (almost) as good in the indifferentiability game as the underlying simulator $\mathcal{S}$ that it is build from. This is captured by the following lemma, for which we additionally need the distinguishing advantage for the indifferentiability game for a distinguisher $\mathcal{D}$ and simulator $\mathcal{S}$ defined as

$$\mathbf{Adv}^{\mathrm{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}) = \left|\Pr\left[\mathcal{D}^{H^{\mathtt{h}},\mathtt{h}}(1^\lambda) = 1\right] - \Pr\left[\mathcal{D}^{\mathcal{R},\mathcal{S}^{\mathcal{R}}}(1^\lambda) = 1\right]\right| .$$

Lemma C.4. *Let $H^{\mathtt{h}} : \{0,1\}^* \to \{0,1\}^n$ be a (t_c, t_e)-iterative hash function and $\mathcal{R}$ a random oracle. Let $\mathcal{S}$ be a simulator that exports the same interface as $\mathtt{h} : \{0,1\}^k \times \{0,1\}^d \to \{0,1\}^s$. Let simulator $\mathcal{S}_*$ be constructed as in Construction C.3 from simulator $\mathcal{S}$. Then for any distinguisher $\mathcal{D}$ making at most q oracle queries there exists distinguisher $\mathcal{D}'$ such that*

$$\mathbf{Adv}^{\mathit{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}_*}(\mathcal{D}) \leq 23q \cdot \mathbf{Adv}^{\mathit{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + 2 \cdot \frac{6(q^2 + (q_{\mathtt{h}} + q_{\mathcal{R}} \cdot \ell)^2) + (q + q_{\mathtt{h}} + q_{\mathcal{R}} \cdot \ell)(|\mathcal{IV}| + |\mathcal{C}|)}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}}$$

Moreover, simulator $\mathcal{S}_*$ has run time $t_{\mathcal{S}_*} \in \mathcal{O}(q \cdot t_{\mathcal{S}} \cdot t_e)$ and makes at most $q_{\mathcal{S}_*} \in \mathcal{O}(q \cdot q_{\mathcal{S}})$ many queries to the random oracle. Distinguisher $\mathcal{D}'$ runs in time $t_{\mathcal{D}} t_{\mathcal{S}_*}$ and makes at most $q+1$ oracle queries. Value $\ell \cdot d$ denotes an upper bound on the length of random oracle queries under function pad *by distinguisher $\mathcal{D}$.*

Proof. We want to upper bound the indifferentiability advantage of a distinguisher $\mathcal{D}$ for simulator $\mathcal{S}_*$:

$$\mathbf{Adv}^{\mathrm{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}_*}(\mathcal{D}) = \left|\Pr\left[\mathcal{D}^{H^{\mathtt{h}},\mathtt{h}}(1^\lambda) = 1\right] - \Pr\left[\mathcal{D}^{\mathcal{R},\mathcal{S}^{\mathcal{R}}_*}(1^\lambda) = 1\right]\right| .$$

For the proof we use a game based approach (similar to the the indifferentiability proofs in [16]) starting from experiment $\Pr\left[\mathcal{D}^{\mathcal{R},\mathcal{S}^{\mathcal{R}}_*}(1^\lambda)=1\right]$ in GAME_1 until we reach the target experiment $\Pr\left[\mathcal{D}^{H^{\mathtt{h}},\mathtt{h}}(1^\lambda)=1\right]$ in GAME_4 summing up the distinguishing probabilities in the individual game hops.

GAME_1. We start with the original security game:

$$\Pr[\,\text{GAME}_1\,] = \Pr\left[\mathcal{D}^{\mathcal{R},\mathcal{S}^{\mathcal{R}}_*}(1^\lambda)=1\right]$$

GAME_2. This game is as the previous game, but for a slightly changed simulator. The new simulator $\mathcal{S}_0$ works as simulator $\mathcal{S}_*$ but looks for conditions that might be exploited by a distinguisher and deliberately fails in such a situation. That is, $\mathcal{S}_0$ fails if one of the following failure conditions occur:

Condition B_1: Simulator $\mathcal{S}_*$ generates an output value as $\mathcal{M}[m,x] \leftarrow \mathtt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})$ such that $\mathtt{g}(\mathcal{M}[m,x]) \neq \mathcal{R}(\mathtt{extract}(\mathfrak{newG}))$.

Condition B_2: Simulator $\mathcal{S}_*$ generates an output value $\mathcal{M}[m,x]$ such that there exists $(m',x') \neq (m,x)$ for which $\mathtt{g}(\mathcal{M}[m',x']) = \mathtt{g}(\mathcal{M}[m,x])$.

Condition B_3: Simulator $\mathcal{S}_*$ generates an output value $\mathcal{M}[m,x]$ such that $\mathcal{M}[m,x] \in \mathcal{IV} \cup \mathcal{C}$.

Condition B_4: The simulator keeps an additional list $\mathcal{L}$ of all queries to it. When on a new query (m,x) a new partial graph $\mathfrak{newG}$ is generated it tests for all earlier queries $(m',x') \in \mathcal{L}$ whether $\mathfrak{newG}$ can be extended by (m',x'). If any query is found, the simulator fails.

Condition B_5: The simulator keeps an additional list $\mathcal{L}$ of all queries to it. When a new output $\mathcal{M}[m,x]$ is chosen the simulator checks if there has been an earlier query $(m',x') \in \mathcal{L}$ such that $\mathcal{M}[m,x] = \mathtt{hp}^{-1}(x')$ or $\mathcal{M}[m,x] = \mathtt{hmp}^{-1}(m')$. If any query is found, the simulator fails.

Let GAME_2 be the event that distinguisher $\mathcal{D}$ outputs one in this setting, i.e.,

$$\Pr[\,\text{GAME}_2\,] = \Pr\left[\mathcal{D}^{\mathcal{R},\mathcal{S}^{\mathcal{R}}_0}(1^\lambda)=1\right] .$$

The responses between the distinguisher in GAME_1 and GAME_2 can only differ, if the simulator reaches one of the failure conditions. For the difference between games GAME_1 and GAME_2 it holds that

$$|\Pr[\,\text{GAME}_2\,] - \Pr[\,\text{GAME}_1\,]| \leq \Pr\left[\bigcup_{i=1}^{5} B_i \text{ hold for any of the queries}\right]$$

We consider the failure conditions in turn. The first failure condition B_1 corresponds to the underlying simulator $\mathcal{S}^{\mathcal{R}}$ failing on properly simulating the responses for a query sequence. Simulator $\mathcal{S}_0$ only generates output values as $\mathcal{M}[m,x] \leftarrow \mathtt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})$ if for the newly generated partial graph $\mathfrak{newG}$ it holds that $\mathtt{extract}(\mathfrak{newG})$ extracts a message M. The simulator is then called on all queries in the partial graph in correct order (by executing $\mathtt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})$).

Assume that the answer computed using simulator $\mathcal{S}^{\mathcal{R}}$ is inconsistent with $\mathcal{R}(M)$, i.e.,

$$\mathtt{g}(\mathtt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})) \neq \mathcal{R}(M) .$$

Then we can build distinguisher $\mathcal{D}'$ against the indifferentiability of $H^{\mathtt{h}}$ with simulator $\mathcal{S}$. Distinguisher $\mathcal{D}'$ guesses index $i \in \{1,\ldots q\}$ and runs distinguisher $\mathcal{D}$ passing on random oracle queries to its random oracle. The first $i-1$ queries to the simulator distinguisher $\mathcal{D}'$ answers by simulating simulator $\mathcal{S}_*$ and in particular simulating the execution of $\mathtt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})$. Here, it runs an internal simulation of $\mathcal{S}$ using its oracle $\mathcal{R}$ to answer random oracle queries. On the i-th query $\mathcal{D}'$ distinguishes between a non-result and a result query. If it is a non-result query, it stops and outputs 1 with probability 1/2. If it is a result query it computes $y \leftarrow \mathtt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})$, this time using its simulator oracle in the computation. It then validates that $\mathtt{g}(y) = \mathcal{R}(\mathtt{extract}(\mathfrak{newG}))$. If this is the case it outputs 0, else it outputs 1. By construction we have, that

$$\Pr[\,B_1 \text{ holds for any of the queries}\,] \leq q \cdot \mathbf{Adv}^{\text{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') .$$

Distinguisher $\mathcal{D}'$ runs in time $t_{\mathcal{D}}t_{\mathcal{S}_*}$ and makes at most $q+1$ oracle queries.

For event B_2 note that if outputs of the simulator are chosen as $\texttt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})$ (line 12) then by failure condition B_1 it holds that

$$\mathfrak{g}(\texttt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})) = \mathcal{R}(\texttt{extract}(\mathfrak{newG})) \ .$$

Now, assume that there exist two partial graphs $\mathfrak{pg}$ and $\mathfrak{pg}'$ in $\mathcal{PG}$ such that $\texttt{extract}(\mathfrak{pg}) = \texttt{extract}(\mathfrak{pg}')$. Then by definition of iterative hash functions it holds that $\mathfrak{pg}$ and $\mathfrak{pg}'$ are isomorphic. We now show that this cannot be the case. Assume that on a fresh query (m,x) a new partial graph $\mathfrak{newG}$ is generated which is isomorphic to an existing graph $\mathfrak{pg} \in \mathcal{PG}$. By construction, the final $\mathfrak{h}$-node in $\mathfrak{newG}$ gets as input values (m,x) if $\mathfrak{newG}$ is evaluated as $\texttt{EVAL}^{\mathcal{S}_0}(\mathfrak{newG})$. Similarly, $\mathfrak{pg}$ has, relative to $\mathcal{S}_0$, values $(m',x') \neq (m,x)$ as input to its final $\mathfrak{h}$-node input. This, however, directly implies that the two graphs cannot be isomorphic.

This directly yields that, if not B_1, then a collision in values generated as $\mathfrak{g}(\texttt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG}))$ can occur with probability at most $q^2 \cdot 2^{-n+1}$ by the birthday bound. If, on the other hand, the output value is not generated as $\texttt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})$ but chosen uniformly at random then the probability of a collision can be upper bound using the min-entropy of a uniformly random variable under function $\mathfrak{g}$. Let U_s denote random variable that is distributed uniformly in $\{0,1\}^s$. Then the probability of a collision under $\mathfrak{g}$ is upper bound by $q^2 \cdot 2^{-\mathrm{H}_\infty(\mathfrak{g}(U_s))+1}$. As $\mathrm{H}_\infty(\mathfrak{g}(U_s)) \leq n$ we have that

$$\Pr[\, B_2 \text{ holds for any of the queries} | \neg B_1 \,] \leq \frac{q^2}{2^{\mathrm{H}_\infty(\mathfrak{g}(U_s))-1}}$$

Failure condition B_3 intuitively says that the simulator never generates an output value that is in set $\mathcal{IV}$ of initialization vectors or in set $\mathcal{C}$ of constants. The output value $\mathcal{M}[m,x]$ is either generated as $\texttt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})$ (line 12) or chosen uniformly at random from $\{0,1\}^s$ (line 14). If $\mathcal{M}[m,x]$ is chosen uniformly at random the probability is at most $(|\mathcal{IV}| + |\mathcal{C}|) \cdot 2^{-s}$ that it is in set $\mathcal{IV} \cup \mathcal{C}$. Taking the union bound over q queries, this yields that the probability is less than $q(|\mathcal{IV}| + |\mathcal{C}|) \cdot 2^{-s}$ that at least one output value is in set $\mathcal{IV} \cup \mathcal{C}$ (given that all q output values were chosen uniformly at random from $\{0,1\}^s$).

If, on the other hand, for a query (m,x) to simulator $\mathcal{S}_0$ a new partial graph $\mathfrak{newG}$ then the output is generated as $\texttt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})$. Assuming event B_1 does not occur, it holds that

$$\mathfrak{g}(\texttt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})) = \mathcal{R}(\texttt{extract}(\mathfrak{newG}))$$

and by the previous discussion we have that no two partial graphs $\mathfrak{pg}$ and $\mathfrak{pg}'$ can be in $\mathcal{PG}$ such that $\texttt{extract}(\mathfrak{pg}) = \texttt{extract}(\mathfrak{pg}')$. Thus, we can upper bound the probability that a single ouput value $\texttt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})$ is in $\mathcal{IV} \cup \mathcal{C}$ by $(|\mathcal{IV}| + |\mathcal{C}|) \cdot 2^{-n}$, assuming that event B_1 does not occur.

Taking a union bound yields

$$\Pr[\, B_3 \text{ holds for any of the queries} | \neg B_1 \,] \leq \frac{q(|\mathcal{IV}| + |\mathcal{C}|)}{2^{\min(s,n)}} \ .$$

For event B_4 consider that, assuming $\neg B_1$, all partial graphs have a different value on their outbound edge, that is:

$$\forall(\mathfrak{pg}, \mathfrak{pg}') \in \mathcal{PG} \times \mathcal{PG} : \mathfrak{pg}.\mathsf{y} \neq \mathfrak{pg}'.\mathsf{y} \vee \mathfrak{pg} = \mathfrak{pg}'$$

For a partial graph $\mathfrak{pg}$ to be extendable by an earlier query $(m',x') \in \mathcal{L}$ it must hold that $\mathfrak{hp}^{-1}(x') = \mathfrak{pg}.\mathsf{y}$.[6] As partial graph $\mathfrak{pg}$ was generated only after query (m',x') was queried to the simulator by the distinguisher we can bound the probability of guessing value x' such that a later query to the underlying simulator $\mathcal{S}^{\mathcal{R}}$ yields value $\mathfrak{hp}^{-1}(x')$ with the birthday bound as $q^2 \cdot 2^{-n+1}$, again using the fact that unless B_1 occurs we have that

$$\mathfrak{g}(\texttt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})) = \mathcal{R}(\texttt{extract}(\mathfrak{newG})) \ .$$

[6] Note that the condition $\mathfrak{hmp}^{-1}(m') = \mathfrak{pg}.\mathsf{y}$ individually is not sufficient to allow for a graph to be extended which is why we can ignore it here.

Thus,

$$\Pr[\, B_4 \text{ holds for any of the queries} | \neg B_1 \,] \leq \frac{q^2}{2^{n-1}}$$

Finally, we can estimate the probability of event B_5 similar to the probability of event B_3 by noting that $\mathcal{L}$ can at any point hold at most q entries. Thus, instead of factor $(|\mathcal{IV}| + |\mathcal{C}|)$ as in B_3 we now get a factor of $2q$:

$$\Pr[\, B_5 \text{ holds for any of the queries} | \neg B_1 \,] \leq \frac{2q^2}{2^{\min(s,n)}}$$

Putting it all together (via a union bound) we have that

$$\begin{aligned}
|\Pr[\,\mathrm{GAME}_2\,] - \Pr[\,\mathrm{GAME}_1\,]| &\leq \Pr\left[\bigcup_{i=1}^{5} B_i \text{ hold for any of the queries}\right] \\
&\leq 5\Pr[\,B_1\,] + \Pr\left[\bigcup_{i=2}^{5} B_i \text{ hold for any of the queries} | \neg B_1\right] \\
&\leq 5q \cdot \mathbf{Adv}^{\mathrm{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + \frac{q^2}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))-1}} + \\
&\qquad \frac{q(|\mathcal{IV}| + |\mathcal{C}|)}{2^{\min(s,n)}} + \frac{q^2}{2^{n-1}} + \frac{2q^2}{2^{\min(s,n)}} \\
&\leq 5q \cdot \mathbf{Adv}^{\mathrm{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + \frac{6q^2 + q(|\mathcal{IV}| + |\mathcal{C}|)}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}} \qquad (1)
\end{aligned}$$

GAME_3. We now change the left oracle (i.e., the random oracle) such that the left oracle is always consistent with the right oracle. That is, instead of the random oracle we now give the distinguisher access to $H^{\mathcal{S}_0}$, that is, the iterative hash construction with the simulator $\mathcal{S}_0$ as oracle. Let GAME_3 be the event that distinguisher $\mathcal{D}$ outputs one in this setting, i.e.,

$$\Pr[\,\mathrm{GAME}_3\,] = \Pr\left[\,\mathcal{D}^{H^{\mathcal{S}_0},\mathcal{S}_0^{\mathcal{R}}}(1^\lambda) = 1\,\right] .$$

We will show that a distinguisher can only detect a difference in the view of GAME_2 and GAME_3 if the simulator $\mathcal{S}_0$ fails in at least one of the two games. In other words we show that in GAME_2 the responses of the simulator are always consistent with the random oracle, unless it explicitly fails.

If the distinguisher wants to distinguish between games GAME_2 and GAME_3 it must find a difference in the responses of the left or right oracle among the two games. The right oracle is the same in both games. The random oracle in game GAME_2 is exchanged for construction $H^{\mathcal{S}_0}$ in game GAME_3. In a first step we show that the left oracle individually does not allow a distinguisher to distinguish the two games.

Simulator $\mathcal{S}_0$ generates its output either as a uniformly random string or via a call to $\mathtt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{new}\mathfrak{G})$. By failure condition B_1 we have that, unless the simulator fails explicitly, the outputs of the simulator are unique and moreover distributed as $\mathtt{g}(U_s)$ where U_s is a random variable uniformly distributed in $\{0,1\}^s$. Thus, a distinguisher $\mathcal{D}$ distinguishing between construction $H^{\mathcal{S}_0}$ and a random oracle can directly be turned into a distinguisher between $H^{\mathtt{h}}$ and a random oracle (without any loss in run-time nor success probability). Thus the advantage of $\mathcal{D}$ is upper bound by $\mathbf{Adv}^{\mathrm{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}')$, that is, the indifferentiability advantage of a distinguisher $\mathcal{D}'$ which runs $\mathcal{D}$ using only its left oracle and outputs whatever $\mathcal{D}$ outputs.

It remains to show that the combined view of the distinguisher does not change. For this note, that in game GAME_3 the responses of the left and right oracle are always consistent. Thus, the distinguisher can only distinguish the two games if it finds inconsistencies between the left and right oracle in game GAME_2. In the following we show that such inconsistencies are only possible if the simulator explicitly fails.

Let us denote by a triple (m, x, y) a call by the distinguisher $\mathcal{D}$ to simulator $\mathcal{S}_0$ where $y := \mathcal{S}_0(m, x)$. Let $Q := (m_i, x_i, y_i)_{1 \leq i \leq q}$ be a sequence of such triples. Then we denote by $\mathcal{PG}(Q)$ the set of potential partial graphs that can be build from queries (and answers) in Q for iterative hash construction $H^{\mathtt{h}}$.

Note that this set may be potentially of infinite size; if the simulator does, however, not fail it will bet at most of size q. The following claim captures that a distinguisher cannot find inconsistencies unless simulator $\mathcal{S}_0$ explicitly fails.

Claim C.5. *In game* GAME_2*, unless simulator* $\mathcal{S}_0$ *explicitly fails, distinguisher* $\mathcal{D}$ *never queries simulator* $\mathcal{S}_0$ *on a sequence* $Q := (m_i, x_i, y_i)_{1 \le i \le q}$ *such that*

$$|\mathcal{PG}(Q)| > q \quad \vee \\ \left(\exists\, \mathfrak{pg} \in \mathcal{PG}(Q) : \mathtt{extract}(\mathfrak{pg}) \neq \bot \quad \wedge \quad \mathcal{R}(\mathtt{extract}(\mathfrak{pg})) \neq \mathfrak{g}(\mathtt{EVAL}^{\mathcal{M}}(\mathfrak{pg}))\right)$$

where $\mathcal{M}$ *is the internal table kept by simulator* $\mathcal{S}_0$ *after processing all queries in* Q*. By* $\mathtt{EVAL}^{\mathcal{M}}(\mathfrak{pg})$ *we denote the execution of algorithm* EVAL *with table* $\mathcal{M}$ *as oracle. If for a query by* EVAL *is not in table* $\mathcal{M}$ *we set the result of computation* $\mathtt{EVAL}^{\mathcal{M}}(\mathfrak{pg})$ *to a value such that*

$$\mathcal{R}(\mathtt{extract}(\mathfrak{pg})) \neq \mathfrak{g}(\mathtt{EVAL}^{\mathcal{M}}(\mathfrak{pg}))\ .$$

Proof. Let us prove the first condition, i.e., $|\mathcal{PG}(Q)| \le q$. In fact, we will not only prove that $|\mathcal{PG}(Q)| \le q$ but that the set $\mathcal{PG}$ as maintained by the simulator $\mathcal{S}_0$ is equivalent to set $\mathcal{PG}(Q)$ unless the simulator explicitly fails. Assume this is not the case. Then as $\mathcal{D}$ makes at most q queries there must exist $i \in \{1, \ldots, q\}$ such that before the i-th query $|\mathcal{PG}(Q_{|_{1,\ldots,i-1}})| < i$ and after the i-th query $|\mathcal{PG}(Q_{|_{1,\ldots,i}})| > i$, that is, the i-th query added at least two partial graphs. Let us first show that the simulator will internally never generate two new partial graphs on any query (that is $\mathfrak{newG}$ is overwritten at least once in line 7 or line 10). For simulator $\mathcal{S}_0$ to generate two partial graphs on a new query (m, x) it must hold that

$$\exists \mathfrak{pg}, \mathfrak{pg}' \in \mathcal{PG} \times \mathcal{PG} : \mathfrak{pg} \neq \mathfrak{pg}' \wedge \mathfrak{pg}.\mathsf{y} = \mathfrak{pg}'.\mathsf{y}\ .$$

By failure condition B_3 this cannot happen if the simulator does not fail explicitly.

Clearly, the set of partial graphs constructed by the simulator is only a subset of $\mathcal{PG}(Q)$ as the simulator only generates partial graphs for queries coming in "correct order". Failure condition B_4, however, directly tells us that earlier queries can never be used to extend later generated partial graphs. Finally, failure conditions B_3 and B_5 tell us that no partial graphs can be used to replace initial queries (B_3) nor can a newly generated partial graph be combined with any of the existing partial graphs (B_5). This proves, that unless the simulator fails explicitly, with each new query at most one new graph is added to $\mathcal{PG}(Q)$. Furthermore, this shows that $\mathcal{PG}(Q)$ is exactly the same as set $\mathcal{PG}$ as constructed by the simulator. Thus, by failure condition B_2 the random oracle must be consistent with answers given by the simulator. ◊

With this we have shown that unless the simulator explicitly fails the view of the distinguisher is in both games equivalent. To complete the game hop we assume that the longest random oracle (left oracle) query by the distinguisher consists of messages such that under pad at most ℓ d-bit blocks are generated. Note that in game GAME_3 these will result in at most ℓ simulator $\mathcal{S}_0$ queries as the left oracle is implemented as $H^{\mathcal{S}_0}$. If we denote by $q_{\mathcal{R}}$ the number of random oracle (left oracle) queries by the distinguisher and by $q_{\mathfrak{h}}$ the number of simulator (right oracle) queries, then we have that

$$|\Pr[\,\mathrm{GAME}_3\,] - \Pr[\,\mathrm{GAME}_2\,]| \le \mathbf{Adv}^{\mathrm{indiff}}_{H^{\mathfrak{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + \Pr[\,\mathcal{S}_0 \text{ fails in } \mathrm{GAME}_2\,] + \Pr[\,\mathcal{S}_0 \text{ fails in } \mathrm{GAME}_3\,]$$

With Equation (1) and noting the difference in the number of $\mathfrak{h}$-queries in games GAME_2 and GAME_3 due to indirect $\mathfrak{h}$-evaluations in game GAME_3 we have:

$$\begin{aligned}
|\Pr[\,\mathrm{GAME}_3\,] - \Pr[\,\mathrm{GAME}_2\,]| &= \mathbf{Adv}^{\mathrm{indiff}}_{H^{\mathfrak{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + \\
&\quad \left(5q \cdot \mathbf{Adv}^{\mathrm{indiff}}_{H^{\mathfrak{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + \frac{6q^2 + q(|\mathcal{IV}| + |\mathcal{C}|)}{2^{\mathrm{H}_\infty(\mathfrak{g}(U_s))}}\right) + \\
&\quad \left(5q \cdot \mathbf{Adv}^{\mathrm{indiff}}_{H^{\mathfrak{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + \frac{6(q_{\mathfrak{h}} + q_{\mathcal{R}} \cdot \ell)^2 + (q_{\mathfrak{h}} + q_{\mathcal{R}} \cdot \ell)(|\mathcal{IV}| + |\mathcal{C}|)}{2^{\mathrm{H}_\infty(\mathfrak{g}(U_s))}}\right) \\
&= 11q \cdot \mathbf{Adv}^{\mathrm{indiff}}_{H^{\mathfrak{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + \\
&\quad \frac{6(q^2 + (q_{\mathfrak{h}} + q_{\mathcal{R}} \cdot \ell)^2) + (q + q_{\mathfrak{h}} + q_{\mathcal{R}} \cdot \ell)(|\mathcal{IV}| + |\mathcal{C}|)}{2^{\mathrm{H}_\infty(\mathfrak{g}(U_s))}}
\end{aligned}$$

GAME_4. In GAME_4 we make the game independent of the random oracle. That is, we now change the simulator to always choose output values for new queries as a uniformly random string in $\{0,1\}^s$. Furthermore, we remove any failure conditions from the simulator. This yields simulator $\mathcal{S}_1$ which effectively simulates a fixed length random oracle via lazy sampling. Let GAME_4 be the event that distinguisher $\mathcal{D}$ outputs one in this setting, i.e.,

$$\Pr[\,\text{GAME}_4\,] = \Pr\left[\,\mathcal{D}^{H^{\mathcal{S}_1},\mathcal{S}_1}(1^\lambda) = 1\,\right] \ .$$

It holds that a distinguisher can only differentiate between games GAME_3 and GAME_4 if

- In game GAME_3, simulator $\mathcal{S}_0$ explicitly fails
- In game GAME_4, simulator $\mathcal{S}_1$ reaches a state in which simulator $\mathcal{S}_0$ would have explicitly failed.
- Distinguisher $\mathcal{D}$ can distinguish between uniformly random values in $\{0,1\}^s$ and values generated as $\texttt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})$

For the last condition note that, unless a simulator explicitly fails we have that

$$\mathfrak{g}(\texttt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})) = \mathcal{R}(\texttt{extract}(\mathfrak{newG})) \ .$$

Assume distinguisher $\mathcal{D}$ can distinguish between values generated as $g^{-1}(\mathcal{R}(\texttt{extract}(\mathfrak{newG})))$ and a uniform value in $\{0,1\}^s$. Then we can build distinguisher $\mathcal{D}'$ against the indifferentiability of $H^{\mathfrak{h}}$ with $\mathcal{S}$. Distinguisher $\mathcal{D}'$ simply runs $\mathcal{D}$ only using its right oracle and outputs whatever $\mathcal{D}$ outputs. Then we have that the distinguishing probability of $\mathcal{D}$ is upper bound by $\mathbf{Adv}^{\text{indiff}}_{H^{\mathfrak{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}')$.

Putting it all together we have that

$$\begin{aligned}
|\Pr[\,\text{GAME}_4\,] - \Pr[\,\text{GAME}_3\,]| &\leq \mathbf{Adv}^{\text{indiff}}_{H^{\mathfrak{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + \Pr[\,\mathcal{S}_0 \text{ fails in } \text{GAME}_3\,] + \\
&\qquad \Pr[\,\mathcal{S}_1 \text{ reaches a failure condition}\,] \\
&= \mathbf{Adv}^{\text{indiff}}_{H^{\mathfrak{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + 22q \cdot \mathbf{Adv}^{\text{indiff}}_{H^{\mathfrak{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + \\
&\qquad 2 \cdot \frac{6(q^2 + (q_{\mathfrak{h}} + q_{\mathcal{R}} \cdot \ell)^2) + (q + q_{\mathfrak{h}} + q_{\mathcal{R}} \cdot \ell)(|\mathcal{IV}| + |\mathcal{C}|)}{2^{\mathrm{H}_\infty(\mathfrak{g}(U_s))}}
\end{aligned}$$

We can now complete the proof of the theorem as we have reached the target view. Note that simulator $\mathcal{S}_1$ effectively implements a fixed length random oracle. Thus:

$$\left|\Pr\left[\,\mathcal{D}^{H^{\mathfrak{h}},\mathfrak{h}}(1^\lambda) = 1\,\right] - \Pr\left[\,\mathcal{D}^{H^{\mathcal{S}_1},\mathcal{S}_1}(1^\lambda) = 1\,\right]\right| = 0$$

and hence

$$\begin{aligned}
\mathbf{Adv}^{\text{indiff}}_{H^{\mathfrak{h}},\mathcal{R},\mathcal{S}_*}(\mathcal{D}) &\leq 23q \cdot \mathbf{Adv}^{\text{indiff}}_{H^{\mathfrak{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + \\
&\qquad 2 \cdot \frac{6(q^2 + (q_{\mathfrak{h}} + q_{\mathcal{R}} \cdot \ell)^2) + (q + q_{\mathfrak{h}} + q_{\mathcal{R}} \cdot \ell)(|\mathcal{IV}| + |\mathcal{C}|)}{2^{\mathrm{H}_\infty(\mathfrak{g}(U_s))}}
\end{aligned}$$

For the run-time of simulator $\mathcal{S}_*$ note that at any time there are at most q graphs in $\mathcal{PG}$. As the test of whether a partial graph can be extended as well as whether the query is an initial query can be done in constant time we have that up-to line 11 the run-time is bound by $\mathcal{O}(q)$. Algorithm `extract` runs in time t_e. The generation of the output value involves either an evaluation of the graph (which has at most $3q$ nodes; per query an $\mathfrak{h}$-node and two preprocessing nodes are added) or the generation of a random value. This can be done in time $\mathcal{O}(q \cdot t_{\mathcal{S}} \cdot t_e)$ noting that the evaluation of the graph can be done in linear time and involves q calls to simulator $\mathcal{S}^{\mathcal{R}}$. □

C.3 Proof of Theorem 5.1

With Lemma C.1 and C.4 we can now prove Theorem 5.1. A proof sketch for the asymptotic setting can be found in the main part in Section 5. Let us restate Theorem 5.1 form page 18, this time in a concrete setting.

Theorem 5.1. *Let $H^{\mathtt{h}} : \{0,1\}^* \to \{0,1\}^n$ be a (t_c, t_e)-iterative hash function and $\mathcal{R}$ a random oracle. Let game $G \in \mathcal{LG}$ be any functionality respecting game that is $(t_{\mathcal{A}^*}, q_{\mathcal{A}^*}, \epsilon_G, \epsilon_{\mathsf{bad}})$-*unsplittable *for $H^{\mathtt{h}}$ and let $\mathcal{A}_1, \ldots, \mathcal{A}_m$ be an adversary. Then, for any indifferentiability simulator $\mathcal{S}$ there exists adversary $\mathcal{B}_1, \ldots, \mathcal{B}_m$ and distinguisher $\mathcal{D}$ such that for all values y*

$$\Pr\left[G^{H^{\mathtt{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m} \Rightarrow y\right] \leq \Pr\left[G^{\mathcal{R},\mathcal{B}_1,\ldots,\mathcal{B}_m} \Rightarrow y\right] + \epsilon_G + \epsilon_{\mathsf{bad}} + 23q_{\mathcal{D}} \cdot \mathbf{Adv}^{indiff}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}) + 2 \cdot \frac{6(q_{\mathcal{D}}^2 + (q_{\mathcal{D},\mathtt{h}} + q_{\mathcal{D},\mathcal{R}} \cdot \ell)^2) + (q_{\mathcal{D}} + q_{\mathcal{D},\mathtt{h}} + q_{\mathcal{D},\mathcal{R}} \cdot \ell)(|\mathcal{IV}| + |\mathcal{C}|)}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}}$$

Moreover,

$$t_{\mathcal{B}_i} \leq \mathcal{O}\left(t_{\mathcal{A}_i^*} + q_{\mathcal{A}_i^*} \cdot q_{\mathcal{D}} \cdot {t_G^*}^2\right) \qquad t_{\mathcal{D}} \leq \mathcal{O}\left((q_{\mathcal{D}} \cdot t_{\mathcal{S}} \cdot t_e)(m + t_G^* + \sum_{i=1}^{m} q_{G,i} \cdot t_{\mathcal{A}_i^*})\right)$$

$$q_{\mathcal{B}_i} \leq \mathcal{O}\left(q_{\mathcal{A}_i^*}(q_{\mathcal{D}} \cdot q_{\mathcal{S}} + t_{\mathcal{S}})\right) \qquad q_{\mathcal{D}} \leq q_{G,0} + \sum_{i=1}^{m} q_{G,i} \cdot t_{\mathcal{A}_i^*} + 1$$

$$\ell \leq \frac{t_{\mathcal{D}}}{d} \qquad t_G^* \leq \mathcal{O}\left(t_G + \sum_{i=1}^{m} q_{G,i} \cdot t_{\mathcal{A}_i^*} \cdot q_{\mathcal{D}} \cdot q_{\mathcal{S}}\right)$$

where $t_{\mathcal{B}_i}, t_{\mathcal{D}}, t_{\mathcal{S}}$ are bounds on the run-times of $\mathcal{B}_i, \mathcal{D}, \mathcal{S}$ and $q_{\mathcal{B}_i}, q_{\mathcal{S}}$ are bounds on the number of oracle queries by procedures $\mathcal{B}_i, \mathcal{S}$. Values $t_{\mathcal{A}_i^*}$ and $q_{\mathcal{A}_i^*}$ are the bounds due to game G being unsplittable *for $H^{\mathtt{h}}$. Values $q_{\mathcal{D},\mathtt{h}}$ and $q_{\mathcal{D},\mathcal{R}}$ denote upper bounds on the number of $\mathtt{h}$-queries and $\mathcal{R}$-queries by $\mathcal{D}$. Values $q_{G,0}$ and $q_{G,i}$ denote upper bounds on the number of queries by game G to the honest interface of the hash functionality and to the i-th adversarial procedure, respectively. Value t_G is an upper bound on the run-time of game $G^{H^{\mathtt{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m}$. Value t_e denotes the runtime of procedure* `extract` *for hash function $H^{\mathtt{h}}$.*

Proof. As game $G^{H^{\mathtt{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m}$ is an $(t_{\mathcal{A}^*}, q_{\mathcal{A}^*}, \epsilon_G, \epsilon_{\mathsf{bad}})$-*unsplittable* for hash function $H^{\mathtt{h}}$ there exists, by definition, adversaries $\mathcal{A}_1^*, \ldots, \mathcal{A}_m^*$ such that for all values y

$$\Pr\left[G^{H^{\mathtt{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m} \Rightarrow y\right] \leq \Pr\left[G^{H^{\mathtt{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*} \Rightarrow y\right] + \epsilon_G$$

and that during game $G^{H_r^{\mathtt{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*}$ bad queries occur only with probability less than ϵ_{bad}. In the rest of the proof we are going to construct adversaries $\mathcal{B}_1, \ldots, \mathcal{B}_m$ such that for all values y

$$\Pr\left[G^{H^{\mathtt{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*} \Rightarrow y\right] \leq \Pr\left[G^{\mathcal{R},\mathcal{B}_1,\ldots,\mathcal{B}_m} \Rightarrow y\right] + \epsilon_{\mathsf{bad}} + 23q_{\mathcal{D}} \cdot \mathbf{Adv}^{\mathrm{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + 2 \cdot \frac{6(q_{\mathcal{D}}^2 + (q_{\mathcal{D},\mathtt{h}} + q_{\mathcal{D},\mathcal{R}} \cdot \ell)^2) + (q_{\mathcal{D}} + q_{\mathcal{D},\mathtt{h}} + q_{\mathcal{D},\mathcal{R}} \cdot \ell)(|\mathcal{IV}| + |\mathcal{C}|)}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}}$$

We will set adversaries $\mathcal{B}_1, \ldots, \mathcal{B}_i$ as

$$\mathcal{B}_i := \mathcal{A}_i^{*\mathcal{S}_d^{(i)}}$$

where $\mathcal{S}_d^{(i)}$ will be an instance of a deterministic simulator $\mathcal{S}_d$. Note that, in contrast to adversary $\mathcal{A}_i^*$, adversary $\mathcal{B}_i$ is used in a game with a random oracle instead of an iterative hash function. The simulator is used to simulate the adversarial interface (i.e., the compression function) which is expected to exist by procedure $\mathcal{A}_i^*$.

Constructing Simulator $\mathcal{S}_d^{(i)}$. We start with a simulator $\mathcal{S}$ that exports the same interface as $\mathtt{h} : \{0,1\}^k \times \{0,1\}^d \to \{0,1\}^s$ and that runs in time $t_{\mathcal{S}}$. From this simulator we are now going to construct simulator $\mathcal{S}_d^{(i)}$ via several intermediate steps that are summarized in Table 1.

In a first step, let $\mathcal{S}_*$ be the simulator constructed from $\mathcal{S}$ according to Construction C.3. By Lemma C.4 we have that for all distinguishers $\mathcal{D}$ there exists distinguisher $\mathcal{D}'$ such that

$$\mathbf{Adv}^{\mathrm{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}_*}(\mathcal{D}) \leq 23q_{\mathcal{D}} \cdot \mathbf{Adv}^{\mathrm{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + 2 \cdot \frac{6(q_{\mathcal{D}}^2 + (q_{\mathcal{D},\mathtt{h}} + q_{\mathcal{R}} \cdot \ell)^2) + (q_{\mathcal{D}} + q_{\mathcal{D},\mathtt{h}} + q_{\mathcal{D},\mathcal{R}} \cdot \ell)(|\mathcal{IV}| + |\mathcal{C}|)}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}}$$

Simulator	Description
$\mathcal{S}$	The basic simulator guaranteed to exist due to hash functionality $H_r^{\mathtt{h}}$ being indifferentiable from a random oracle.
$\mathcal{S}_*$	The simulator constructed according to Construction C.3 with $\mathcal{S}$ as underlying simulator.
$\mathcal{S}_*^d$	The derandomized version of $\mathcal{S}_*$ with derandomization with respect to adversarial procedures $\mathcal{A}_1^*, \ldots, \mathcal{A}_m^*$.
$\mathcal{S}_d$	As $\mathcal{S}_*^d$ but stateless.

Table 1: Simulators used in proof of Theorem 5.1

The run-time of our new simulator $\mathcal{S}_*$ is upper bound by $\mathcal{O}(q_{\mathcal{D}} \cdot t_{\mathcal{S}} \cdot t_e)$ where $q_{\mathcal{D}}$ is an upper bound on the number of oracle queries by distinguisher $\mathcal{D}$. Values $q_{\mathtt{h},\mathcal{D}}, q_{\mathcal{R},\mathcal{D}}$ denote an upper bound on the number of $\mathtt{h}$-queries (resp. $\mathcal{R}$-queries) by $\mathcal{D}$, that is, $q_{\mathcal{D}} \geq q_{\mathtt{h},\mathcal{D}} + q_{\mathcal{R},\mathcal{D}}$. Value t_e denotes the run-time of algorithm `extract` and U_s is a random variable that is uniformly distributed in $\{0,1\}^s$.

In the next step we are going to derandomize this simulator using the techniques developed in Lemma C.1. We set $\mathcal{S}_*^d$ to be the derandomized simulator constructed from simulator $\mathcal{S}_*$ using Lemma C.1. For this we choose the bound on the run-time of the distinguisher in Lemma C.1 as t_G^* which is an upper bound on the run-time of game $G^{\mathcal{R},\mathcal{A}_1^{*\mathcal{S}_*},\ldots,\mathcal{A}_m^{*\mathcal{S}_*}}$:

$$t_G^* \leq \mathcal{O}\left(t_G + \sum_{i=1}^{m} q_{G,i} \cdot t_{\mathcal{A}_i^*} \cdot q_{\mathcal{D}} \cdot q_{\mathcal{S}}\right) .$$

Value t_G denotes the run-time of the original game $G^{H^{\mathtt{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m}$ with adversarial procedures $\mathcal{A}_1, \ldots, \mathcal{A}_m$. Value $q_{G,i}$ denotes a bound on the number of calls to the i-th adversarial procedure by game G. Note that by this choice of run-time bound for the derandomization it holds, in particular, that for any of the adversarial procedures $\mathcal{A}_i^*$ it is not detectable whether it is given oracle access to $\mathcal{S}_*$ or $\mathcal{S}_*^d$.

From simulator $\mathcal{S}_*^d$ we now construct a simulator that is not just deterministic but also stateless with respect to non-result queries as well as stateless for the evaluation of $\mathtt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})$. Note that $\mathcal{S}_*^d$ distinguishes between result and non-result queries (see lines 12 and 14 in Figure 13). Non-result queries are answered by randomly chosen values (where the randomness now, due to derandomization, is extracted from the random oracle), while result queries are answered by computing the entire partial graph with the underlying simulator $\mathcal{S}^{\mathcal{R}}$ (which also takes extracted randomness from the random oracle instead of real randomness). We will now change the simulator such that its answers are independent from the simulator's internal state.

Simulator $\mathcal{S}_d$ works as simulator $\mathcal{S}_*^d$ but generates values for non-result queries in the following way. To answer non-result query (m, x) simulator $\mathcal{S}_d$ computes

$$y := \mathcal{R}(r \| m \| x) \qquad \text{with} \qquad r := \mathcal{R}(0^{t_G+1}) .$$

Note that by computing value r via a random oracle query to 0^{t_G+1} we generate a value that is independent of the bit stream used to derandomize the simulator (cp. Lemma C.1). Further note that the probability of any of the adversarial procedures $\mathcal{A}_i^*$ or game G of guessing value r is upper bounded by 2^{-n} as they cannot have queried the oracle on that message due to the runtime restriction.

For the generation of answers to result queries note that the underlying simulator $\mathcal{S}$ is always reset before the execution of $\mathtt{EVAL}^{\mathcal{S}^{\mathcal{R}}}(\mathfrak{newG})$. To ensure that the result does not depend on state we assume that `EVAL` always evaluates the same (or isomorphic) graph in the same order[7], and secondly we deterministically generate the randomness for simulator $\mathcal{S}$ by the same approach as for our other derandomization steps. We choose the randomness for the simulator from a bit stream generated as

$$\mathcal{R}(01^{t_G^*+1}) \| \mathcal{R}(01^{t_G^*+2}) \| \ldots$$

[7]For example, algorithm `EVAL` can, whenever it has a choice, choose to process the node with the lexicographically smallest input first.

Note that neither step changes the success probability for a distinguisher $\mathcal{D}$ and thus it still holds for simulator $\mathcal{S}_d$ that

$$\mathbf{Adv}^{\text{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}_d}(\mathcal{D}) \leq 23q_{\mathcal{D}} \cdot \mathbf{Adv}^{\text{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + 2 \cdot \frac{6(q_{\mathcal{D}}^2 + (q_{\mathcal{D},\mathtt{h}} + q_{\mathcal{D},\mathcal{R}} \cdot \ell)^2) + (q_{\mathcal{D}} + q_{\mathcal{D},\mathtt{h}} + q_{\mathcal{D},\mathcal{R}} \cdot \ell)(|\mathcal{IV}| + |\mathcal{C}|)}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}} \tag{2}$$

Construction Adversary $\mathcal{B}_i$. We can now construct adversary $\mathcal{B}_i$ as

$$\mathcal{B}_i := \mathcal{A}_i^{*\,\mathcal{S}_d^{(i)}}$$

where $\mathcal{S}_d^{(i)}$ denotes an independent copy of simulator $\mathcal{S}_d$. By construction, we know that for all non-result queries the simulation is guaranteed to be consistent, as independent instances of simulator $\mathcal{S}_d$ give the same answer to the same query. Further, by construction, during game $G^{\mathcal{R},\mathcal{B}_1,\ldots,\mathcal{B}_m}$ bad queries happen only with probability at most ϵ_{bad}. Thus, for result queries, all queries in the corresponding partial graph are with probability at least $1 - \epsilon_{\mathsf{bad}}$ queried by the same adversarial procedure in correct order. As this allows the procedure's copy of the simulator $\mathcal{S}_d$ to recognize the partial graph, it follows that also result queries can be answered consistently by the respective copy of simulator $\mathcal{S}_d$. For the asymptotic setting, where by $\approx$ we denote negligibly-close, we can complete the proof noting that for all values y it holds

$$\begin{aligned}\Pr\left[G^{\mathcal{R},\mathcal{B}_1,\ldots,\mathcal{B}_m} \Rightarrow y\right] \approx \\ \Pr\left[G^{\mathcal{R},\mathcal{A}_1^{*\,\mathcal{S}_d^{(1)}},\ldots,\mathcal{A}_m^{*\,\mathcal{S}_d^{(m)}}} \Rightarrow y\right] \approx \Pr\left[G^{\mathcal{R},\mathcal{A}_1^{*\,\mathcal{S}_d},\ldots,\mathcal{A}_m^{*\,\mathcal{S}_d}} \Rightarrow y\right] \approx \Pr\left[G^{H^{\mathtt{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*} \Rightarrow y\right] \\ \approx \Pr\left[G^{H^{\mathtt{h}},\mathcal{A}_1,\ldots,\mathcal{A}_m} \Rightarrow y\right]\end{aligned}$$

For concrete security, the remainder of the proof is readily established by applying Theorem 6.1 in [30] (Theorem 4 in the proceedings version [31]), that is, the composition theorem by Ristenpart et al. for reset indifferentiable hash functions. Note that our simulator is with probability $1 - \epsilon_{\mathsf{bad}}$ equivalent to a resettable simulator where a reset call precedes any adversarial procedure call and which is the form of simulator needed in the proof of Theorem 6.1 [30]. We thus have

$$\Pr\left[G^{H^{\mathtt{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*}\right] \leq \Pr\left[G^{\mathcal{R},\mathcal{B}_1,\ldots,\mathcal{B}_m}\right] + \mathbf{Adv}^{\text{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}_d}(\mathcal{D}) + \epsilon_{\mathsf{bad}} \ .$$

with

$$t_{\mathcal{B}_i} \leq t_{\mathcal{A}_i^*} + q_{\mathcal{A}_i^*} \cdot t_{\mathcal{S}_d} \qquad q_{\mathcal{B}_i} \leq q_{\mathcal{A}_i^*} \cdot q_{\mathcal{S}_d} \qquad t_{\mathcal{D}} \leq m + t_G^* + \sum_{i=1}^{m} q_{G,i} \cdot t_{\mathcal{A}_i^*} \qquad q_{\mathcal{D}} \leq q_{G,0} + \sum_{i=1}^{m} q_{G,i} \cdot t_{\mathcal{A}_i^*}$$

where $t_{\mathcal{D}}$ is a bound on the run-time of distinguisher $\mathcal{D}$ and $q_{\mathcal{D}}$ is a bound on the number of oracle queries by $\mathcal{D}$. Value $q_{G,0}$ denotes a bound on the number of queries by G to the honest interface of the functionality and $q_{G,i}$ denotes a bound on the number of calls to the i-th adversarial procedure (see [30] Theorem 6.1). Value $t_{\mathcal{B}_i}$ is an upper bound on the run-time of procedure $\mathcal{B}_i$ and $q_{\mathcal{B}_i}$ an upper bound on the number of random oracle queries by $\mathcal{B}_i$. Value $t_{\mathcal{A}_i^*}$ denotes the run-time of procedure $\mathcal{A}_i^*$ and $q_{\mathcal{A}_i^*}$ a bound on the number of oracle calls to the adversarial interface of $H^{\mathtt{h}}$ by procedure $\mathcal{A}_i^*$. The run-time of simulator $\mathcal{S}_d$ is denoted with $t_{\mathcal{S}_d}$ and $q_{\mathcal{S}_d}$ is a bound on the simulators oracle queries.

Plugging in equation (2) we get that

$$\Pr\left[G^{H^{\mathtt{h}},\mathcal{A}_1^*,\ldots,\mathcal{A}_m^*}\right] \leq \Pr\left[G^{\mathcal{R},\mathcal{B}_1,\ldots,\mathcal{B}_m}\right] + \epsilon_{\mathsf{bad}} + 23q_{\mathcal{D}} \cdot \mathbf{Adv}^{\text{indiff}}_{H^{\mathtt{h}},\mathcal{R},\mathcal{S}}(\mathcal{D}') + 2 \cdot \frac{6(q_{\mathcal{D}}^2 + (q_{\mathcal{D},\mathtt{h}} + q_{\mathcal{D},\mathcal{R}} \cdot \ell)^2) + (q_{\mathcal{D}} + q_{\mathcal{D},\mathtt{h}} + q_{\mathcal{D},\mathcal{R}} \cdot \ell)(|\mathcal{IV}| + |\mathcal{C}|)}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}}$$

It remains to estimate the run-time and query complexity of distinguisher $\mathcal{D}'$ and simulator $\mathcal{S}_d$. By Lemma C.4 we have that $t_{\mathcal{D}'} \leq t_{\mathcal{D}} t_{\mathcal{S}_*} \in \mathcal{O}(t_{\mathcal{D}} \cdot q_{\mathcal{D}} \cdot t_{\mathcal{S}} \cdot t_e)$ where $t_{\mathcal{S}}$ is the run-time of the original simulator $\mathcal{S}$. Distinguisher $\mathcal{D}'$ makes $q_{\mathcal{D}'} = q_{\mathcal{D}} + 1$ oracle queries. Simulator $\mathcal{S}_d$ runs in time of the derandomized version of $\mathcal{S}_*$, that is, $t_{\mathcal{S}_d} \in \mathcal{O}\left(q_{\mathcal{D}} \cdot t_{\mathcal{S}} \cdot t_G^* + q_{\mathcal{D}} \cdot t_{\mathcal{S}}^2\right)$ and as $t_G^* > t_{\mathcal{S}}$ this becomes $t_{\mathcal{S}_d} \in \mathcal{O}\left(q_{\mathcal{D}} \cdot {t_G^*}^2\right)$. Simulator $\mathcal{S}_*$ makes $\mathcal{O}(q_{\mathcal{D}} \cdot q_{\mathcal{S}})$ queries to the random oracle. The derandomized version thus makes $q_{\mathcal{S}_d} \in \mathcal{O}(q_{\mathcal{D}} \cdot q_{\mathcal{S}} + t_{\mathcal{S}})$. For value ℓ note that it is an upper bound on the number of d-bit blocks in $\mathtt{pad}(M)$ where M is the longest message by distinguisher $\mathcal{D}$. We can thus bound ℓ by $t_{\mathcal{D}}/d$. $\square$

D The Non-Adaptive CDA Game

In this section we prove that the non-adaptive CDA game (Figure 1) is *unsplittable* for any iterative hash construction under the same assumptions as Ristenpart et al. [31] showed that for the non-adaptive CDA game composition works as in the indifferentiability framework for hash functions of the form $\mathtt{g}(f^{\mathtt{h}}(\cdot))$. Note that the difference to our proof is, that in the NMAC-case $\mathtt{g}$ is assumed to be an ideal function independent of $\mathtt{h}$. The version due to Ristenpart et al. is, thus, not applicable, for example, for Chop-MD functions such as SHA-2 or Keccak.

Furthermore, we give a second and simpler proof and show that under an independent set of assumptions —yet to some extend less restrictive assumptions than the assumptions by Ristenpart et al.— the CDA game is *unsplittable* for a large class of iterative hash constructions.

We begin by recalling the basic definitions.

Public-Key Encryption. A public-key encryption scheme $\mathcal{AE} := (\mathsf{KGen}, \mathcal{E}, \mathcal{D})$ consists of three efficient algorithms: a key generation algorithm KGen that given the security parameter generates a keypair (pk, sk), an encryption algorithm $\mathcal{E}$ that, being given a message m, randomness r, and the public key pk, outputs a ciphertext c, and the decryption algorithm $\mathcal{D}$ that, given a ciphertext c and secret key sk, outputs a plaintext message or a distinguished symbol $\perp$.

CDA Security. The CDA game (depicted in Figure 1) captures the security of public-key encryption schemes where the randomness used to encrypt may not be sufficiently random after all, i.e., it may not have sufficient min-entropy [6]. For the remainder of this and the next section we denote by $\omega > 0$ the size of messages and by $\rho > 0$ the size of randomness for encryption scheme $\mathcal{E}$. In the CDA-game adversary $\mathcal{A}_1$ implements a so called (μ, ν)-$\mathtt{mmr}$-source which is a probabilistic algorithm that outputs a triplet of vectors $(\mathbf{m}_0, \mathbf{m}_1, \mathbf{r})$, each of size ν. Vectors $\mathbf{m}_0$ and $\mathbf{m}_1$ contain messages, that is, each component is of size ω and vector $\mathbf{r}$ corresponds to randomness, that is each component is of size ρ. Furthermore, to exclude trivial attacks, we require that $(\mathbf{m}_b[i], \mathbf{r}[i]) \neq (\mathbf{m}_b[j], \mathbf{r}[j])$ for all $1 \leq i < j \leq \nu$ and all $b \in \{0,1\}$. Finally, we require that components have sufficient min-entropy μ independent of the random oracle, that is for all $1 \leq i \leq \nu$, all $b \in \{0,1\}$, all $r \in \{0,1\}^{\rho}$, and all $m \in \{0,1\}^{\omega}$ it holds that

$$\Pr\left[\, (\mathbf{m}_b[i], \mathbf{r}[i]) = (m, r) | (\mathbf{m_0}, \mathbf{m_1}, \mathbf{r}) \leftarrow \mathcal{A}_1^{\mathcal{R}}(1^{\lambda}), \mathcal{R} \,\right] \leq 2^{-\mu} \ .$$

The advantage of an adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$ in the CDA game where adversary $\mathcal{A}_1$ is a valid (μ, ν)-$\mathtt{mmr}$-source is given as

$$\mathbf{Adv}^{\mathrm{CDA}}_{\mathcal{AE}, H^{\mathtt{h}}}(\mathcal{A}_1, \mathcal{A}_2) := 2 \cdot \Pr\left[\mathrm{CDA}^{H^{\mathtt{h}}, \mathcal{A}_1, \mathcal{A}_2}_{\mathcal{AE}} \Rightarrow \mathsf{true} \right] - 1 \ .$$

D.1 Composition for CDA - An Instructive Example

In the following we are going to prove that composition works for CDA under three simple assumptions. As the second assumption will narrow down the class of iterative hash functions for which CDA is *unsplittable*, this composition theorem is a bit more restricted than the composition theorem we show in Section D.2. Nevertheless, the here presented theorem holds, for example, for Merkle-Damgård like hash functions, and furthermore, makes fewer assumptions on the encryption scheme used in the CDA game. Moreover, the proof is straightforward and we believe quite instructive on how to work with the notion of *unsplittable*-games.

The first assumption is that the the encryption scheme $\mathcal{AE}$ queries the random oracle on a single message m per $\mathcal{E}$ invocation such that m is of the form $m := pk \| \ldots$, that is, m is prepended with the public key pk. Secondly, we assume that if $\mathcal{E}$ is queried on two inputs with the same length, then also its query to the hash function is of the same length. These two requirements are, for example, met by the Randomized-Encrypt-with-Hash scheme [6] and the Encrypt-with-Hash scheme [5].

The third and final assumption is on the iterative hash construction $H^{\mathtt{h}}$. We assume that padding function $\mathtt{pad}$ only adds a postfix to any message, that is, $M = \mathtt{pad}(M)_{|_{0..|M|}}$. Let $m_1 \| \ldots \| m_{\ell} = \mathtt{pad}(M)$ be a padded message consisting of ℓ blocks of size d. We assume that for function $H^{\mathtt{h}}$ it holds that the execution graph $\mathtt{construct}(M)$ has exactly one $\mathtt{h}$-node connected to an $\mathcal{IV}$-node (via an $\mathtt{hp}$-node, and assuming $|\mathcal{IV}| = 1$) and that the second input to that $\mathtt{h}$-node is connected to a message-block-node for

message-block m_1 (via a mp-node). Note that this requirement is met, for example, by Chop-MD or the sponge construction (see Section A.2).

Finally, we need a notion of the probability of guessing the public key. For this, we define the maximum public-key collision probability in the style of [5]. This intuitively captures the probability of guessing a public key as generated by a PKE scheme's KGen algorithm:

$$\mathsf{maxpk}_{\mathcal{AE}} := \max_{w \in \{0,1\}^*} \Pr\left[\, (pk, sk) \leftarrow \mathsf{KGen}(1^\lambda) : pk = w \,\right] \tag{3}$$

Lemma D.1. *Let $\mathcal{AE}$ be a public-key encryption scheme. Let the encryption scheme query its hash functionality $H^{\mathtt{h}}$ on a single message per $\mathcal{E}$ invocation, that message being of the form $pk\|\ldots$. Let adversary $\mathcal{A}_1$ be a valid (μ,ν)-*mmr*-source. Then, the non-adaptive $CDA^{H^{\mathtt{h}},\mathcal{A}_1,\mathcal{A}_2}_{\mathcal{AE}}$ game (cf. Figure 1) is $(t_{\mathcal{A}^*}, q_{\mathcal{A}^*}, \epsilon_G, \epsilon_{\mathsf{bad}})$-*unsplittable *for any iterative hash construction with restrictions as described above. Moreover*

$$t_{\mathcal{A}_i^*} = \mathcal{O}(t_{\mathcal{A}_i} + q_{\mathcal{A}_i}) \qquad \epsilon_{\mathsf{bad}} = 0$$

$$q_{\mathcal{A}_i^*} \leq 2q_{\mathcal{A}_i} \qquad \epsilon_G \leq \mathsf{maxpk}_{\mathcal{AE}} + \frac{(q_{\mathcal{A}_1} + q_{\mathcal{A}_2}) \cdot \nu \cdot q_{\mathcal{E}}}{2^s} + \frac{\nu^2 q_{\mathcal{E}}^2}{2^{s-1}}$$

Values $t_{\mathcal{A}_i}, t_{\mathcal{A}_i^*}$ denote the run-time of $\mathcal{A}_i, \mathcal{A}_i^*$; values $q_{\mathcal{A}_i}, q_{\mathcal{A}_i^*}$ denote the number of oracle queries. Value $q_{\mathcal{E}}$ denotes an upper bound on the number of $\mathtt{h}$ *evaluations on an $H^{\mathtt{h}}$-query by $\mathcal{E}$.*

To ease on notation we assume in the upcoming proof that $|pk| = d$ as well as $k \leq d$, that is, the public key fits exactly the block-length of scheme $H^{\mathtt{h}}$ and the second input to $\mathtt{h} : \{0,1\}^d \times \{0,1\}^k \to \{0,1\}^s$ is at most of length d.

Proof. The outline of the proof is best explained with Figure 14. Game G_1 is the original CDA game. The idea is to exchange the oracle that the adversarial procedures are given access to for an oracle $\mathtt{h}^{\mathtt{h}}_{\mathcal{S}}$ which ensures that no query is bad by slightly changing how queries are forwarded to the actual oracle $\mathtt{h}$. The key point is, that the first stage adversary can only with probability less than $\mathsf{maxpk}_{\mathcal{AE}}$ guess public key pk. Thus, the probability of $\mathcal{A}_1$ querying an $\mathtt{h}$-query which is also (indirectly) queried by the encryption function $\mathcal{E}$ is bound by $\mathsf{maxpk}_{\mathcal{AE}}$ as the very first query in every partial graph will contain the public-key.

Let $\mathtt{h} : \{0,1\}^d \times \{0,1\}^k \to \{0,1\}^s$ be an ideal function. Let $q_{\mathcal{A}_1}$ be the number of oracle queries by $\mathcal{A}_1$ and let $q_{\mathcal{E}}$ (a function of ω and ρ, the message length and randomness length) be a bound on the number of $\mathtt{h}$-nodes in any partial graph constructed as the result of an $H^{\mathtt{h}}$-query by $\mathcal{E}$. Note that there are ν-many entries in vectors $\mathbf{m_b}, \mathbf{r_b}$ and thus ν-oracle queries by $\mathcal{E}$. Let $\mathsf{iv} \in \mathcal{IV}$ be the single initial vector for scheme $H^{\mathtt{h}}$. For each $H^{\mathtt{h}}$-query by $\mathcal{E}$ the single initial query is (iv, pk). The probability of any of the $q_{\mathcal{A}_1}$ oracle queries by $\mathcal{A}_1$ being (iv, pk) is upper bound by $q_{\mathcal{A}_1} \cdot \mathsf{maxpk}_{\mathcal{AE}}$. Assuming $\mathcal{A}_1$ does not query (iv, pk) then we can directly upper bound the probability of a collision of a query by $\mathcal{A}_1$ with an indirect query by $\mathcal{E}$ (due to calling $H^{\mathtt{h}}$) by $q_{\mathcal{A}_1} \cdot \nu \cdot q_{\mathcal{E}} \cdot 2^{-s}$. Thus, we have

$$\Pr[\,\mathcal{A}_1 \text{ } \mathtt{h}\text{-collides with } \mathcal{E}\,] \leq \mathsf{maxpk}_{\mathcal{AE}} + \frac{q_{\mathcal{A}_1} \cdot \nu \cdot q_{\mathcal{E}}}{2^s}$$

Let us now describe the setup of game G_2. For this let $\mathtt{h}_{\mathcal{S}} : \{0,1\}^d \times \{0,1\}^k$ be defined as

$$\mathtt{h}_{\mathcal{S}}(m, x) := \mathtt{h}(x, \mathtt{h}(m, \mathsf{iv}))\ .$$

Note that function $\mathtt{h}_{\mathcal{S}}$ queries a Merkle-Damgård chain of length two with the first query being an initial query. Thus, by only querying $\mathtt{h}_{\mathcal{S}}$ it is impossible to make bad queries. Further note, that as $\mathtt{h}$ is an ideal function, functions $\mathtt{h}_{\mathcal{S}}$ and $\mathtt{h}$ are indistinguishable.

Let $\mathtt{h}^*_{\mathcal{S}} : \{0,1\}^d \times \{0,1\}^k$ be defined as

$$\mathtt{h}^*_{\mathcal{S}}(m, x) := \begin{cases} \mathtt{h}(m, x) & \text{if } (m, x) \text{ belongs to partial graph which contains initial query } (pk, \mathsf{iv}) \\ \mathtt{h}_{\mathcal{S}}(m, x) & \text{else} \end{cases}$$

We have already seen (for example, the extractor in Lemma 3.5 or the simulator in Construction C.3) that it is easy to keep a list of partial graphs and on a new query (m, x) check if a partial graph can be

extended by that query. Function $\mathtt{h}^*_{\mathcal{S}}$ does exactly this, it keeps a list of all partial graphs that begin with query (pk, iv) (as $\mathtt{h}^*_{\mathcal{S}}$ is only used by $\mathcal{A}_2$ we can assume that $\mathtt{h}^*_{\mathcal{S}}$ is initialized with value pk) and on a new query (m, x) it checks, if any of the partial graphs can be extended by that query. If this is the case, then it returns $\mathtt{h}(m, x)$, else it returns $\mathtt{h}_{\mathcal{S}}(m, x)$. Note that the function is not well defined, as a query (m, x) might at an earlier stage not be part of a correct partial graph, but be so at a later point. If this happens, we let $\mathtt{h}^*_{\mathcal{S}}$ explicitly fail and say the adversary wins. We will shortly see how to bound this probability.

We further need to bound the probability of adversary $\mathcal{A}_2$ making query (m, x) to its oracle such that $\mathtt{h}^*_{\mathcal{S}}$ answers with $\mathtt{h}_{\mathcal{S}}$ (that is, the second case) and query (m, x) was (indirectly) queried by $\mathcal{E}$ during some $H^{\mathtt{h}}$ evaluation. Let us call this event "$\mathcal{A}_2$ $\mathtt{h}$-collides with $\mathcal{E}$". In this case it holds for query (m, x) that $\mathtt{hp}^{-1}(x)$ is equal to the result of an $\mathtt{h}$-query (m', x') by $\mathcal{E}$ and $\mathcal{A}_2$ did not make query (m', x'). The probability of guessing $\mathtt{h}(m', x')$ is at most 2^{-s}. However, $\mathcal{A}_2$ gets as input vector $\mathbf{c}$. In the following we argue that even given vector $\mathbf{c}$ the probability of an $\mathtt{h}$-collision between $\mathcal{A}_2$ and $\mathcal{E}$ is small, unless $\mathcal{A}_2$ queries the messages in the same order as $\mathcal{E}$.

It holds that given $H^{\mathtt{h}}(M)$ any $\mathtt{h}$-query in the corresponding execution graph, except for initial queries, have min-entropy at least s bits (also see the proof of Lemma 3.4). As $\mathcal{E}$ does not make $\mathtt{h}$-queries directly this, then, also holds given value $c_i = \mathcal{E}(pk, \mathbf{m_b}[i]; \mathbf{r}[i])$. Thus, value c_i can at most contain information about the result of the final $\mathtt{h}$-query during the computation of $H^{\mathtt{h}}(M_i)$ as well as information about the input to $\mathcal{E}$, that is, $pk, \mathbf{m_b}[i]; \mathbf{r}[i])$. As for $i, j \in \{1, \ldots, \nu\}$ we have that

$$|pk| + |\mathbf{m_b}[i]| + |\mathbf{m_b}[i]| = |pk| + |\mathbf{m_b}[j]| + |\mathbf{m_b}[j]|$$

also, by definition, queries M_i and M_j by $\mathcal{E}$ are of equal length, that is $|M_i| = |M_j|$. Thus, the probability that $\mathtt{h}(m^i_{\mathtt{res}}, x^i_{\mathtt{res}})$, where $(m^i_{\mathtt{res}}, x^i_{\mathtt{res}})$ denotes the result query corresponding to $H^{\mathtt{h}}(M_i)$, appears as second input in any of the $\mathtt{h}$-queries in $H^{\mathtt{h}}(M_j)$ is upper bounded by the probability of having an $\mathtt{h}$-collision within the $q_{\mathcal{E}}$ indirect $\mathtt{h}$-queries by $\mathcal{E}$ during the evaluation of $H^{\mathtt{h}}(M_j)$. Thus, we can safely upper bound the probability that $\mathcal{A}_2$ $\mathtt{h}$-collides with $\mathcal{E}$ by

$$\frac{\nu^2 q_{\mathcal{E}}^2}{2^{s-1}} + \frac{q_{\mathcal{A}_2} \cdot \nu \cdot q_{\mathcal{E}}}{2^s}$$

where the first term corresponds to a collision within the $q_{\mathcal{E}} \times \nu$ indirect queries by $\mathcal{E}$ and the second term corresponds to $\mathcal{A}_2$ guessing an $\mathtt{h}$-query by $\mathcal{E}$. Note that if $\mathcal{A}_2$ does not $\mathtt{h}$-collide with $\mathcal{E}$ then simulator $\mathtt{h}^*_{\mathcal{S}}$ never fails.

We are now almost done. We have established that if $\mathcal{A}_2$ does not $\mathtt{h}$-collides with $\mathcal{E}$ we have that the view of adversary $\mathcal{A}_2$ is consistent with that in game G_1. That is, if it queries a partial graph that was also queried by the encryption scheme, then $\mathtt{h}$ is used directly. Furthermore, if in addition $\mathcal{A}_1$ does not $\mathtt{h}$-collides with $\mathcal{E}$, then if $\mathcal{A}_2$ makes a query (m, x) to its oracle which was also made by $\mathcal{A}_1$ then $\mathtt{h}_{\mathcal{S}}$ is used. Thus:

$$\begin{aligned}|\Pr[G_1 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| &\leq \Pr[\mathcal{A}_1 \text{ } \mathtt{h}\text{-collides with } \mathcal{E}] + \Pr[\mathcal{A}_2 \text{ } \mathtt{h}\text{-collides with } \mathcal{E}] \\ &\leq \mathsf{maxpk}_{\mathcal{AE}} + \frac{q_{\mathcal{A}_1} \cdot \nu \cdot q_{\mathcal{E}}}{2^s} + \frac{\nu^2 q_{\mathcal{E}}^2}{2^{s-1}} + \frac{q_{\mathcal{A}_2} \cdot \nu \cdot q_{\mathcal{E}}}{2^s}\end{aligned}$$

Moreover, bad queries in G_2 can, by construction, not occur. Adversary $\mathcal{A}_1^{\mathtt{h}_{\mathcal{S}}}$ with $\mathtt{h}_{\mathcal{S}}$ as oracle runs in time $t_{\mathcal{A}_1}$ but now makes two oracle queries per $\mathtt{h}$ query by $\mathcal{A}_1^{\mathtt{h}}$. Adversary $\mathcal{A}_2^{\mathtt{h}^*_{\mathcal{S}}}$ with $\mathtt{h}^*_{\mathcal{S}}$ needs to keep an additional list of partial graphs. There can be at most $q_{\mathcal{A}_2}$ partial graphs and the checks if a query belongs to any one of them can be done in linear time. Thus $\mathcal{A}_2^{\mathtt{h}^*_{\mathcal{S}}}$ runs in time $\mathcal{O}(t_{\mathcal{A}_2} + q_{\mathcal{A}_2})$. □

D.2 Reproving the CDA Composition Theorem due to Ristenpart et al. [31]

In this section we are reproving the composition result for the CDA game due to Ristenpart et al. [31]. The two key differences between our version and the one due to Ristenpart et al. is that i) our proof is a bit simpler, and, furthermore it holds for any iterative hash function and not only for functions of the NMAC type, that is, functions that can be decomposed as $\mathtt{g}(f^{\mathtt{h}}(\cdot))$ where $\mathtt{g}$ and $\mathtt{h}$ are two independent ideal functions.

game $\mathbf{G_1}$	**game $\mathbf{G_2}$**
$b \leftarrow \{0,1\}$	$b \leftarrow \{0,1\}$
$(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$	$(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$
$(\mathbf{m_0}, \mathbf{m_1}, \mathbf{r}) \leftarrow \mathcal{A}_1^{\mathrm{h}}(1^\lambda)$	$(\mathbf{m_0}, \mathbf{m_1}, \mathbf{r}) \leftarrow \mathcal{A}_1^{\mathrm{h}_S}(1^\lambda)$
$\mathbf{c} \leftarrow \mathcal{E}^{H^{\mathrm{h}}}(pk, \mathbf{m_b}; \mathbf{r})$	$\mathbf{c} \leftarrow \mathcal{E}^{H^{\mathrm{h}}}(pk, \mathbf{m_b}; \mathbf{r})$
$b' \leftarrow \mathcal{A}_2^{\mathrm{h}}(pk, \mathbf{c})$	$b' \leftarrow \mathcal{A}_2^{\mathrm{h}_S^*}(pk, \mathbf{c})$
else return $(b = b')$	else return $(b = b')$

Figure 14: Games for Proof of Lemma D.1

game $\text{IND-SIM}^{\mathcal{A},H^{\mathrm{h}}}_{\mathcal{AE},\mathcal{S}}$	**procedure** $\mathtt{RoS}(m, r)$
$b \leftarrow \{0,1\}$	If $b = 1$ then return $\mathcal{E}^{H^{\mathrm{h}}}(pk, m; r)$
$(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$	else return $\mathcal{S}^{H^{\mathrm{h}}}(pk, \lvert m\rvert)$
$b' \leftarrow \mathcal{A}^{\mathtt{RoS},\mathrm{h}}(pk)$	
return $(b = b')$	

Figure 15: Game IND-SIM

IND-SIM Security. To prove that CDA is *unsplittable* for any hash construction (Lemma 4.2) we need the additional notion of IND-SIM security defined by Ristenpart et al. in [31]. Intuitively IND-SIM states that an adversary cannot distinguish between real and simulated encryptions, where the simulator only gets the public key and message length as input: this is captured via the IND-SIM game (Figure 15). Here, the only limititation is that the adversary may not repeat queries to its RoS oracle. We define the IND-SIM advantage of an adversary $\mathcal{A}$ as

$$\mathbf{Adv}^{\text{IND-SIM}}_{\mathcal{AE},\mathcal{S}}(\mathcal{A}) := 2 \cdot \Pr\left[\,\text{IND-SIM}^{\mathcal{A}}_{\mathcal{AE},\mathcal{S}} \Rightarrow \mathsf{true}\,\right] - 1$$

Do note that the adversary chooses both message and randomness for the encryption and that without additional randomness this security goal is unachievable. It is, however, achieveable (for example, by the Randomized-Encrypt-with-Hash scheme [6] and the Encrypt-with-Hash scheme [5]) if the adversary does not make any random oracle calls [31]. We can, naturally, weaken this restriction to require only that the adversary does not make any random oracle calls which would also be made by the *deterministic* algorithm $\mathcal{E}$ for any of the adversaries RoS-queries.

Full Proof of Lemma 4.2. We prove Lemma 4.2 under similar assumptions as do Ristenpart et al. [31] in their analysis. The first assumption is, that the encryption scheme is IND-SIM secure in the ROM if the IND-SIM adversary does not make random oracle queries. The second assumption is that the encryption scheme uses the public key, as well as its input (i.e., the to be encrypted message and provided randomness) in its single query to the hash oracle per $\mathcal{E}$-invocation. Third, we assume the probability of guessing the public key as generated by KGen to be small. Finally we assume that adversary $\mathcal{A}_1$ is a valid mmr-source. Let us restate the lemma:

Lemma 4.2 (restated). *Let $\mathcal{AE}$ be a public-key encryption scheme. Let the encryption scheme query its hash construction H^{h} on a single message per $\mathcal{E}$ invocation, that message including (an encoding of) the public key and the input to $\mathcal{E}$. Let adversary $\mathcal{A}_1$ be a valid (μ,ν)-mmr-source. Then, for any encryption simulator $\mathcal{S}$ there exists IND-SIM-adversary $\mathcal{B}$ such that the non-adaptive $CDA^{H^{\mathrm{h}},\mathcal{A}_1,\mathcal{A}_2}_{\mathcal{AE}}$ game (cf. Figure 1) is $(t_{\mathcal{A}^*}, q_{\mathcal{A}^*}, \epsilon_G, \epsilon_{\mathsf{bad}})$-*unsplittable *for any hash construction. Moreover,*

$$\epsilon_G \le 3q_{\mathcal{A}_1} \cdot \mathsf{maxpk}_{\mathcal{AE}} + \frac{3(q_{\mathcal{A}_1} + q_{\mathcal{A}_2}) \cdot \nu + 2q_{\mathcal{B}} + 6q_{\mathcal{B}}^2}{2^s} + \frac{3q_{\mathcal{A}_2} \cdot \nu}{2^\mu} + \frac{6\nu}{2^{\mathrm{H}_\infty(\mathsf{g}(U_s))}} + 2 \cdot \mathbf{Adv}^{\mathit{IND\text{-}SIM}}_{\mathcal{AE},\mathcal{R},\mathcal{S}}(\mathcal{B})$$

$$\epsilon_{\mathsf{bad}} = 0 \qquad t_{\mathcal{A}_i^*} \le t_{\mathcal{A}_i} \qquad q_{\mathcal{A}_i^*} \le 2q_{\mathcal{A}_i} \qquad q_{\mathcal{B}} \le q_{\mathcal{A}_1} + q_{\mathcal{A}_2} + \nu \cdot \max(q_{\mathcal{E}}, q_{\mathcal{S}})$$

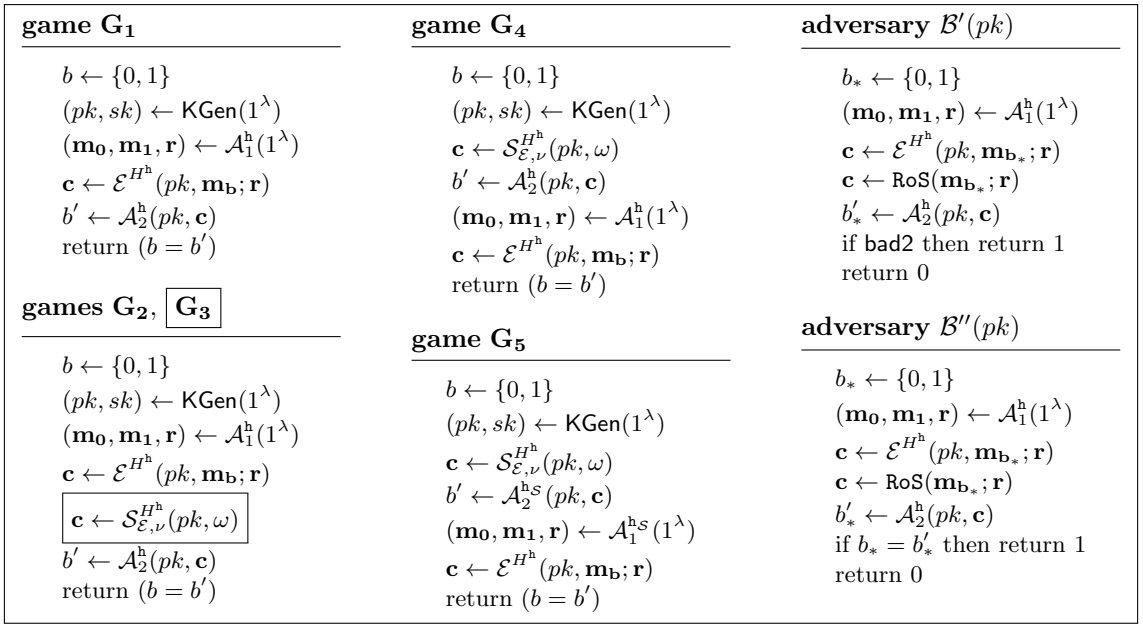

Figure 16: Games for Proof of Lemma 4.2

Adversary $\mathcal{B}$ runs in time of game $CDA_{\mathcal{AE}}^{H^{\mathtt{h}},\mathcal{A}_1,\mathcal{A}_2}$, makes no random oracle queries and ν many RoS*-queries. Value $t_{\mathcal{A}_i}$ (resp. $q_{\mathcal{A}_i}$) denotes the run-time of procedure $\mathcal{A}_i$ (resp. the number of oracle queries by adversary $\mathcal{A}_i$). Value U_s denotes a random variable that is uniformly distributed in $\{0,1\}^s$. Values $q_\mathcal{E}, q_\mathcal{S}$ are upper bounds on the number of (indirect)* h*-queries during the computation of $\mathcal{E}^{H^{\mathtt{h}}}$ (resp. $\mathcal{S}^{H^{\mathtt{h}}}$).*

The proof itself is somewhat similar to the proof of Theorem 9.1 by Ristenpart et al. [31]. For easier notation, we assume that the encryption scheme embeds the public key pk such that during its sole query to the hash construction $H^{\mathtt{h}}$ there occurs a query $\mathtt{h}(pk, x)$, that is, the public key is not spread over various message blocks.

Proof of Lemma 4.2. The games needed for the proof are depicted in Figure 16. Game G_1 (depicted in Figure 16 is the original CDA game.

$$\Pr\left[CDA_{\mathcal{AE}}^{H^{\mathtt{h}},\mathcal{A}_1,\mathcal{A}_2} \Rightarrow \mathsf{true}\right] = \Pr[G_1 \Rightarrow \mathsf{true}]$$

For game G_2 (here the boxed statement in Figure 16 is not executed) we fail if one of two conditions occur. Let $Q_\mathcal{E}$ denote the queries by encryption scheme $\mathcal{E}$ to hash functionality $H^{\mathtt{h}}$. Let bad1 (resp. bad2) be the event that adversary $\mathcal{A}_1$ (resp. $\mathcal{A}_2$) in the original CDA game computes value $\mathtt{g}(\mathtt{h}(m, x))$ such that

$$\exists M \in Q_\mathcal{E} : H^{\mathtt{h}}(M) = \mathtt{g}(\mathtt{h}(m, x)) \ .$$

As the encryption scheme embeds public key pk in any of its queries and we can bound the probability that any of the $q_{\mathcal{A}_1}$ oracle queries by $\mathcal{A}_1$ contains pk by $q_{\mathcal{A}_1} \cdot \mathsf{maxpk}_{\mathcal{AE}}$. If a single h-query in the computation of $H^{\mathtt{h}}(M)$ for some message M is not queried, we can bound the probability that the corresponding result query is queried by $q_{\mathcal{A}_1} 2^{-s}$. For this note that the output of an h-query has min-entropy s bits and that these s bits are transported through the graph $\mathtt{construct}(M)$ up-to the result query (also see Lemma 3.4). As with probability $1 -_{\mathcal{A}_1} \cdot \mathsf{maxpk}_{\mathcal{AE}}$ value $\mathtt{h}(pk, x)$ not queried we have that

$$\Pr[\mathsf{bad1}] \leq q_{\mathcal{A}_1} \cdot \mathsf{maxpk}_{\mathcal{AE}} + \frac{q_{\mathcal{A}_1} \cdot \nu}{2^s} + \frac{\nu}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}}$$

where by U_s we denote a random variable that is uniformly distributed in $\{0,1\}^s$. For the difference between games G_1 and G_2 then holds.

$$\begin{aligned} |\Pr[G_2 \Rightarrow \mathsf{true}] - \Pr[G_1 \Rightarrow \mathsf{true}]| &\leq \Pr[\mathsf{bad1}] + \Pr[\mathsf{bad2}] \\ &\leq q_{\mathcal{A}_1} \cdot \mathsf{maxpk}_{\mathcal{AE}} + \frac{q_{\mathcal{A}_1} \cdot \nu}{2^s} + \frac{\nu}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}} + \Pr[\mathsf{bad2}] \end{aligned}$$

Let $\mathcal{S}^{H^{\mathtt{h}}}_{\mathcal{E},\nu}$ be an algorithm that on input (pk, ω) runs the encryption simulator $\mathcal{S}_{\mathcal{E}}(pk, \omega)$ ν times outputting the vector of results. Game G_3 is exactly as game G_2 except that after the generation of the ciphertext vector $\mathbf{c}$ the vector is overwritten with simulated ciphertext via a call to $\mathcal{S}^{H^{\mathtt{h}}}_{\mathcal{E},\nu}$.

Now consider adversaries $\mathcal{B}'$ and $\mathcal{B}''$ depicted in Figure 16 against the IND-SIM-notion. Both adversary perfectly simulate game G_2 but for a call to its RoS-oracle, after the generation of the ciphertexts. The only difference between $\mathcal{B}'$ and $\mathcal{B}''$ is that $\mathcal{B}'$ returns 1 in case event bad2 occurs while $\mathcal{B}''$ returns 1 in case $\mathcal{A}_2$ guessed correctly. Note that in both adversaries $\mathbf{c} \leftarrow \mathcal{E}^{H^{\mathtt{h}}}(pk, \mathbf{m}_{\mathbf{b}_*}; \mathbf{r})$ is executed with the result value ignored.[8] This is done solely to be consistent with games G_1 and G_2 and to capture event bad2. Let us by IND-SIM1 denote the game IND-SIM with hidden bit b always set to 1 and where the game outputs the adversary's guess directly (Figure 17). Analogously, we define IND-SIM0 as the game IND-SIM with hidden bit b set to 0 (Figure 18). Then, by construction, we have

$$\Pr\left[\text{IND-SIM1}^{\mathcal{B}'}_{\mathcal{AE}} \Rightarrow \mathsf{true}\right] = \Pr[\,\mathsf{bad2} \text{ during } G_2\,] \quad \text{and} \quad \Pr\left[\text{IND-SIM0}^{\mathcal{B}'}_{\mathcal{AE}} \Rightarrow \mathsf{true}\right] = \Pr[\,\mathsf{bad2} \text{ during } G_3\,]$$

and

$$\Pr\left[\text{IND-SIM1}^{\mathcal{B}''}_{\mathcal{AE}} \Rightarrow \mathsf{true}\right] = \Pr[G_2 \Rightarrow \mathsf{true}] \quad \text{and} \quad \Pr\left[\text{IND-SIM0}^{\mathcal{B}''}_{\mathcal{AE}} \Rightarrow \mathsf{true}\right] = \Pr[G_3 \Rightarrow \mathsf{true}]$$

Let $\mathcal{B}$ be whichever of $\mathcal{B}'$ and $\mathcal{B}''$ achieves a larger advantage. We now further adapt adversary $\mathcal{B}$ such that instead of calling $\mathtt{h}$ on result queries it chooses a random value as result. By this we ensure, that the adversary never queries a complete an $H^{\mathtt{h}}$ execution for any message. In other words, in the random oracle setting the adversary does not make random oracle queries. For this, we set $\mathcal{B}^* := \mathcal{B}^{\mathtt{h}_*}$ where $\mathtt{h}_*$ tracks partial graphs (this can be done as using the extractor from Lemma 3.5) in and returns random (but consistent) values on result queries. Result queries are with respect to the view of $\mathtt{h}_*$. We here give a simplified pseudo-code omitting part of keeping track of partial graphs. For this see the extractor in Lemma 3.5.

Function $\mathtt{h}_*(m, x)$:
  if $\mathcal{M}[m, x] \neq \perp$ **then return** $\mathcal{M}[m, x]$
  if (m, x) result-query **then** $\mathcal{M}[m, x] \leftarrow \{0,1\}^s$
  else $(m, x) \leftarrow \mathtt{h}(m, x)$
  return $\mathcal{M}[m, x]$

By Lemma 3.5 we have that

$$\mathbf{Adv}^{\text{IND-SIM}}_{\mathcal{AE},\mathcal{R},\mathcal{S}}(\mathcal{B}) \leq \mathbf{Adv}^{\text{IND-SIM}}_{\mathcal{AE},\mathcal{R},\mathcal{S}}(\mathcal{B}^*) + \Pr[\,\mathsf{bad1}\,] + \Pr[\,\mathsf{bad2}\,] + \frac{q_{\mathcal{B}} + 3q_{\mathcal{B}}^2}{2^s}$$

where $q_{\mathcal{B}} \leq q_{\mathcal{A}_1} + q_{\mathcal{A}_2} + \nu \cdot \max(q_{\mathcal{E}}, q_{\mathcal{S}_{\mathcal{E}}})$ denotes the number of oracle queries by $\mathcal{B}$ and values $q_{\mathcal{E}}, q_{\mathcal{S}_{\mathcal{E}}}$ denotes an upper bound on the number of (indirect) $\mathtt{h}$-queries during the computation of $\mathcal{E}$ (resp. $\mathcal{S}_{\mathcal{E}}$) on input $pk, \mathbf{m}_{\mathbf{b}}[i]; \mathbf{r}[i]$ (resp. (pk, ω)). Futhermore, adversary $\mathcal{B}^*$ does not make any random oracle queries.

Now we have that

$$\begin{aligned}
|\Pr[G_3 \Rightarrow \mathsf{true}] - \Pr[G_2 \Rightarrow \mathsf{true}]| &\leq 2 \cdot \mathbf{Adv}^{\text{IND-SIM}}_{\mathcal{AE},\mathcal{R},\mathcal{S}}(\mathcal{B}) \\
&\leq 2\left(\mathbf{Adv}^{\text{IND-SIM}}_{\mathcal{AE},\mathcal{R},\mathcal{S}}(\mathcal{B}^*) + \Pr[\,\mathsf{bad1}\,] + \Pr[\,\mathsf{bad2}\,] + \frac{q_{\mathcal{B}} + 3q_{\mathcal{B}}^2}{2^s}\right) \\
&\leq 2 \cdot \mathbf{Adv}^{\text{IND-SIM}}_{\mathcal{AE},\mathcal{R},\mathcal{S}}(\mathcal{B}^*) + 2 \cdot \Pr[\,\mathsf{bad2}\,] + \frac{2q_{\mathcal{B}} + 6q_{\mathcal{B}}^2}{2^s} + \\
&\qquad 2 \cdot \left(q_{\mathcal{A}_1} \cdot \mathsf{maxpk}_{\mathcal{AE}} + \frac{q_{\mathcal{A}_1} \cdot \nu}{2^s} + \frac{\nu}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}}\right)
\end{aligned}$$

Note that in game G_3 we have that the outcome is independent of hidden bit b and thus

$$\Pr[G_3 \Rightarrow \mathsf{true}] = 1/2\ .$$

[8] Value b_* is the bit simulated by adversaries $\mathcal{B}'$ and $\mathcal{B}''$ and should not be confused with the underlying hidden bit b by the IND-SIM-game.

game IND-SIM1$^{\mathcal{A},H^{\mathtt{h}}}_{\mathcal{AE},\mathcal{S}}$	**procedure** $\mathtt{RoS}(m,r)$
$(pk,sk) \leftarrow \mathsf{KGen}(1^\lambda)$ $b' \leftarrow \mathcal{A}^{\mathtt{RoS},\mathtt{h}}(pk)$ return b'	return $\mathcal{E}^{H^{\mathtt{h}}}(pk,m;r)$

Figure 17: Game IND-SIM1

game IND-SIM0$^{\mathcal{A},H^{\mathtt{h}}}_{\mathcal{AE},\mathcal{S}}$	**procedure** $\mathtt{RoS}(m,r)$
$(pk,sk) \leftarrow \mathsf{KGen}(1^\lambda)$ $b' \leftarrow \mathcal{A}^{\mathtt{RoS},\mathtt{h}}(pk)$ return b'	return $\mathcal{S}^{H^{\mathtt{h}}}(pk,\lvert m\rvert)$

Figure 18: Game IND-SIM0

What is left is to bind the probability of event $\mathsf{bad2}$. For this, we move to another game G_4 in which we change the order of adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$. As the input to adversary $\mathcal{A}_2$ only depends on the simulator's output we have that

$$\Pr[G_4 \Rightarrow \mathsf{true}] = \Pr[G_3 \Rightarrow \mathsf{true}] \qquad \text{and} \qquad \Pr[\mathsf{bad2} \text{ during } G_4] = \Pr[\mathsf{bad2} \text{ during } G_3]$$

Encryption algorithm $\mathcal{E}$ on input (pk,m,r) queries its random oracle on a message containing (an encoding of) the public key, message m, and randomness r. Adversary $\mathcal{A}_1$ is a valid (μ,ν)-$\mathtt{mmr}$-source and, thus, the probability of guessing a particular query M to $H^{\mathtt{h}}$ by $\mathcal{E}$ is upper-bounded by $2^{-\mu}$. Thus, for the computation of $H^{\mathtt{h}}(M)$ for some query M by $\mathcal{E}$ the uncertainty about M is mapped to the uncertainty about $\mathtt{h}$-queries. As $\mathtt{h}$ is ideal, we can, thus, similarly assume that during the computation of $H^{\mathtt{h}}(M)$ there occurs an $\mathtt{h}$ query (m,x) such that (m,x) has min-entropy μ-bits. If a single query during the evaluation of graph $\mathtt{construct}(M)$ is not made, we can upper bound the probability of making the corresponding result query by with Lemma 3.4. Thus, we can upper bound the probability of event $\mathsf{bad2}$ by

$$\Pr[\mathsf{bad2} \text{ during } G_4] \leq \frac{q_{\mathcal{A}_2}\cdot\nu}{2^\mu} + \frac{q_{\mathcal{A}_2}\cdot\nu}{2^s} + \frac{\nu}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}}$$

Finally, to bound the probability of a bad query, we move to game G_5. Game G_5 is exactly as game G_4 except that we exchange the access to ideal function $\mathtt{h}$ for adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ by access to a function $\mathtt{h}_{\mathcal{S}}$ which is defined as in the proof of Lemma D.1 as

$$\mathtt{h}_{\mathcal{S}}(m,x) := \mathtt{h}(x,\mathtt{h}(m,\mathsf{iv}))\ .$$

As $\mathtt{h}_{\mathcal{S}}$ is indistinguishable from $\mathtt{h}$ we have that, unless $\mathsf{bad2}$ occurs

$$\Pr[G_5 \Rightarrow \mathsf{true}] = \Pr[G_4 \Rightarrow \mathsf{true}]$$

Putting it all together we have that

$$\begin{aligned}\epsilon_G := \left|\Pr[G_5 \Rightarrow \mathsf{true}] - \Pr\left[CDA^{H^{\mathtt{h}},\mathcal{A}_1,\mathcal{A}_2}_{\mathcal{AE}} \Rightarrow \mathsf{true}\right]\right| &\leq 3\cdot\mathsf{bad1} + 3\cdot\mathsf{bad2} + 2\cdot\mathbf{Adv}^{\text{IND-SIM}}_{\mathcal{AE},\mathcal{R},\mathcal{S}}(\mathcal{B}^*) + \frac{2q_{\mathcal{B}}+6q_{\mathcal{B}}^2}{2^s}\\ &\leq 3q_{\mathcal{A}_1}\cdot\mathsf{maxpk}_{\mathcal{AE}} + \frac{3(q_{\mathcal{A}_1}+q_{\mathcal{A}_2})\cdot\nu + 2q_{\mathcal{B}}+6q_{\mathcal{B}}^2}{2^s} +\\ &\quad \frac{3q_{\mathcal{A}_2}\cdot\nu}{2^\mu} + \frac{6\nu}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}} + 2\cdot\mathbf{Adv}^{\text{IND-SIM}}_{\mathcal{AE},\mathcal{R},\mathcal{S}}(\mathcal{B}^*)\end{aligned}$$

As with $\mathtt{h}_{\mathcal{S}}$ it is impossible to make bad queries we have that $\epsilon_{\mathsf{bad}} = 0$. □

E The Adaptive CDA Game

In the adaptive CDA game [6] (see Figure 20) the first adversary can adaptively generate ciphertexts before it has to output the two message vectors $\mathbf{m_0}, \mathbf{m_1}$ and the randomness vector $\mathbf{r}$. For this, we

game $\text{PK-EXT}^{\mathcal{A},H^{\mathtt{h}}}_{\mathcal{AE}}$	**procedure** $\mathtt{ENC}(m,r)$
$(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$ $pk' \leftarrow \mathcal{A}^{\mathtt{ENC},\mathtt{h}}(1^\lambda)$ return $(pk = pk')$	return $\mathcal{E}^{H^{\mathtt{h}}}(pk, m; r)$

Figure 19: Game PK-EXT

give adversary $\mathcal{A}_1$ access to an oracle ENC, which allows to encrypt messages under the public key, but without having to give $\mathcal{A}_1$ access to the public key (except, of course, what is revealed by ENC-queries).

PK-EXT Security. In order to prove that the adaptive CDA game is *unsplittable* we need an extra assumption on the encryption scheme: namely, given the encryption of a message, it should be infeasible to extract the public key used in the encryption. Bellare et al. [4] define the notion of key indistinguishability (IK-CPA, see Figure 21) for public-key encryption schemes which intuitively captures that no adversary given an encryption can learn anything about the public key used for the encryption. The notion is defined as an indistinguishability notion, where a first-stage adversary gets two distinct public keys and outputs a message. According to some secret bit b this message is encrypted with one of the two public keys and given to a second stage adversary that has to guess b. Note that this notion cannot be fulfilled by any PKE scheme if the adversary is allowed to chose the randomness used in the encryption. Here the second-stage adversary can simply, on its own, recompute the ciphertext for both public keys and compare the outcome to its input.

We propose a weaker notion that can be met even if the adversary chooses the randomness used by the encryption scheme. We define the notion of PK-EXT (short for public-key extractability) for public-key encryption schemes. Game $\text{PK-EXT}^{\mathcal{A},H^{\mathtt{h}}}_{\mathcal{AE}}$ is shown in Figure 19. An adversary can make multiple queries to an encryption oracle and then has to output a guess for the public key that was used for the encryptions. We define the advantage of an adversary $\mathcal{A}$ by

$$\mathbf{Adv}^{\text{PK-EXT}}_{\mathcal{AE}}(\mathcal{A}) := \Pr\left[\text{PK-EXT}^{\mathcal{A}}_{\mathcal{AE}} \Rightarrow \mathsf{true}\right].$$

Note that this property is a natural strengthening of the property that public keys output by the key generation algorithm should not be guesseable. However, it is still quite a weak property as it only requires that super-logarithmically many bits of the public key have to remain hidden. In Appendix F we prove that our new notion is met by the REwH1 scheme [6] if the underlying PKE scheme is IK-CPA secure. Examples of IK-CPA-secure schemes are, for example, the El Gamal or the Cramer-Shoup schemes [17]. We can further show that in case the adversary cannot specify the randomness, then PK-EXT is directly implied by IK-CPA.

The adaptive CDA game is unsplittable. The proof in the adaptive setting is essentially equivalent to the proof in the non-adaptive setting (see Appendix D.2) once we have bound the probability of event bad1, that is, the probability that adversary $\mathcal{A}_1$ computes value $\mathsf{g}(\mathtt{h}(m,x)) = H^{\mathtt{h}}(M)$ for some message M send to oracle $H^{\mathtt{h}}$ by $\mathcal{E}$. Let us first restate the lemma:

Lemma E.1. *Let the setup be as in Lemma 4.2 and let the encryption scheme be PK-EXT-secure. Let the encoding of the public key in messages of $\mathcal{E}$ to the hash functionality be invertible, that is given a query to the hash functionality the public key can be extracted. Then, for any encryption simulator $\mathcal{S}$ there exists IND-SIM-adversary $\mathcal{B}$ and PK-EXT-adversary $\mathcal{C}$ such that the adaptive $aCDA^{H^{\mathtt{h}},\mathcal{A}_1,\mathcal{A}_2}_{\mathcal{AE}}$ game (cf. Figure 20) is $(t_{\mathcal{A}^*}, q_{\mathcal{A}^*}, \epsilon_G, \epsilon_{\mathsf{bad}})$*-unsplittable *for any hash construction. Moreover,*

$$\epsilon_G \leq \frac{3 \cdot q_{\mathcal{A}_1,\mathtt{h}} + 9 \cdot q^2_{\mathcal{A}_1,\mathtt{h}} + 3 \cdot q_{\mathcal{A}_2} \cdot \nu + 2q_{\mathcal{B}} + 6q^2_{\mathcal{B}}}{2^s} + \frac{3 \cdot q_{\mathcal{A}_2} \cdot \nu}{2^\mu} + \frac{3 \cdot \nu + 3}{2^{\mathrm{H}_\infty(\mathsf{g}(U_s))}} + 2 \cdot \mathbf{Adv}^{\mathit{IND\text{-}SIM}}_{\mathcal{AE},\mathcal{R},\mathcal{S}}(\mathcal{B}) + 2q_{\mathcal{A}_1,\mathtt{h}} \cdot \mathbf{Adv}^{\mathit{PK\text{-}EXT}}_{\mathcal{AE}}(\mathcal{C})$$

$$\epsilon_{\mathsf{bad}} = 0 \qquad t_{\mathcal{A}^*_i} \leq t_{\mathcal{A}_i} \qquad q_{\mathcal{A}^*_i} \leq 2q_{\mathcal{A}_i} \qquad q_{\mathcal{B}} \leq q_{\mathcal{A}_1} + q_{\mathcal{A}_2} + \nu \cdot \max(q_{\mathcal{E}}, q_{\mathcal{S}})$$

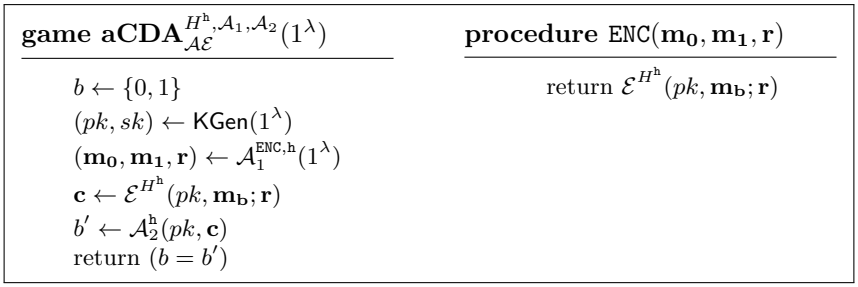

Figure 20: The adaptive CDA game

Adversary $\mathcal{B}$ runs in time of game $aCDA^{H^{\mathtt{h}},\mathcal{A}_1,\mathcal{A}_2}_{\mathcal{AE}}$, makes no random oracle queries and ν many RoS*-queries. Value $t_{\mathcal{A}_i}$ denotes the run-time of procedure $\mathcal{A}_i$. Values $q_{\mathcal{A}_1,\mathtt{h}}$ (resp. $q_{\mathcal{A}_2}$) denotes the number of* h*-queries by adversary $\mathcal{A}_1$ (resp. $\mathcal{A}_2$). Value U_s denotes a random variable that is uniformly distributed in $\{0,1\}^s$. Adversary $\mathcal{C}$ runs in time of game $aCDA^{H^{\mathtt{h}},\mathcal{A}_1,\mathcal{A}_2}_{\mathcal{AE}}$ and makes at most $q_{\mathcal{A}_1,\mathtt{h}}$* h*-queries and at most $q_{\mathcal{A}_1,\mathtt{ENC}}$* ENC*-queries. Values $q_{\mathcal{E}}, q_{\mathcal{S}}$ are upper bounds on the number of (indirect)* h*-queries during the computation of $\mathcal{E}^{H^{\mathtt{h}}}$ (resp. $\mathcal{S}^{H^{\mathtt{h}}}$).*

Proof. Let $Q_{\mathcal{E}}$ denote the queries by encryption scheme $\mathcal{E}$ to hash functionality $H^{\mathtt{h}}$. We want to bind the probability of event bad1, that is the event that adversary $\mathcal{A}_1$ computes value $\mathtt{g}(\mathtt{h}(m,x))$ such that

$$\exists M \in Q_{\mathcal{E}} : H^{\mathtt{h}}(M) = \mathtt{g}(\mathtt{h}(m,x)) \ .$$

In case bad1 occurs, then we know that as an encoding of pk is embedded into all messages $M \in Q_{\mathcal{E}}$ also adversary $\mathcal{A}_1$ must query its oracle on this encoding of the public key. This allows us to construct adversary $\mathcal{C}$ against the PK-EXT security of the encryption scheme $\mathcal{AE}$. Adversary $\mathcal{C}$ chooses a random bit b_* and runs adversary $\mathcal{A}_1$. It answers queries $(\mathbf{m_0}, \mathbf{m_1}, \mathbf{r})$ by adversary $\mathcal{A}_1$ to oracle ENC using its own ENC-oracle as $\mathtt{ENC}(\mathbf{m_{b_*}}, r)$. Note that, by construction, $\mathcal{C}$ perfectly simulates the first stage of an adaptive CDA game. Furthermore, as $\mathcal{C}$ sees all of $\mathcal{A}$'s queries it can reconstruct possible corresponding messages M, that is, it reconstructs all possible partial graphs pg and uses $\mathtt{extract}(\mathtt{pg})$ to reconstruct the corresponding messages. By Lemma 3.5 algorithm $\mathcal{C}$ fails in reconstructing all possible messages with probability at most

$$\frac{q_{\mathcal{A}_1,\mathtt{h}} + 3q^2_{\mathcal{A}_1,\mathtt{h}}}{2^s} + \frac{1}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}}$$

where $q_{\mathcal{A}_1,\mathtt{h}}$ denotes the number of h-evaluations by $\mathcal{A}_1$ and U_s is a random variable uniformly distributed in $\{0,1\}^s$. Adversary $\mathcal{C}$ now guesses an index $1 \leq i \leq q_{\mathcal{A}_1}$ and extracts a public key from message M that corresponds to one of the at most 2 partial graphs that are generated on the i-th query by $\mathcal{A}_1$ (that is a partial graph was generated; also see Lemma 3.5). By construction, adversary $\mathcal{C}$ wins, if the corresponding message M is also queried to $H^{\mathtt{h}}$ by $\mathcal{E}$.

Thus, we can bound the probability of event bad1 by:

$$\Pr[\,\mathsf{bad1}\,] \leq \frac{q_{\mathcal{A}_1,\mathtt{h}} + 3q^2_{\mathcal{A}_1,\mathtt{h}}}{2^s} + \frac{1}{2^{\mathrm{H}_\infty(\mathtt{g}(U_s))}} + 2q_{\mathcal{A}_1} \cdot \mathbf{Adv}^{\text{PK-EXT}}_{\mathcal{AE}}(\mathcal{C})$$

The remainder of the proof follows with the proof for the non-adaptive case. □

F Public-Key Extractability (PK-EXT) for PKE Schemes

Bellare et al. [4] define the notion of key indistinguishability (IK-CPA, see Figure 21) for PKE schemes which intuitively captures that no adversary can tell with which key, out of a known set, a ciphertext was encrypted. The notion is formalized as an indistinguishability experiment, where two keys are generated and given to the first-stage adversary $\mathcal{A}_1$ which outputs a target message. According to a secret bit b the message is encrypted with one of the two keys and the ciphertext is given to the second-stage adversary $\mathcal{A}_2$ which has to output a guess for b (note that as there is no restriction on the state shared by the

game $\text{IK-CPA}^{\mathcal{A}_1,\mathcal{A}_2}_{\mathcal{AE}}(1^\lambda)$

$b \leftarrow \{0,1\}$
$(pk_0, sk_0) \leftarrow \mathsf{KGen}(1^\lambda)$
$(pk_1, sk_1) \leftarrow \mathsf{KGen}(1^\lambda)$
$m, st \leftarrow \mathcal{A}_1(pk_0, pk_1)$
$c \leftarrow \mathcal{E}^{H^{\mathrm{h}}}(pk_b, m)$
$b' \leftarrow \mathcal{A}_2^{\mathrm{h}}(c, st)$
return $(b = b')$

Figure 21: Key Indistinguishability under Chosen-Plaintext Attack

two adversaries this is essentially a single-stage notion). The advantage of an adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$ against IK-CPA is defined as

$$\mathbf{Adv}^{\text{IK-CPA}}_{\mathcal{AE}}(\mathcal{A}) := 2 \cdot \Pr\left[\,\text{IK-CPA}^{\mathcal{A}}_{\mathcal{AE}} \Rightarrow \mathsf{true}\,\right] - 1$$

It is easy to see that if the adversary can additionally choose the randomness of the scheme the notion cannot be fullfilled, as adversary $\mathcal{A}_2$ could then simply recompute the ciphertext for both keys and check which one it received.

In this section we show that our notion of PK-EXT-secure PKE schemes is fullfilled by the REwH1 scheme [6], if the underlying scheme is IK-CPA secure. Further, if the adversary cannot choose the randomness then IK-CPA implies PK-EXT for any PKE scheme.

Randomized-Encrypt-with-Hash. The Randomized-Encrypt-with-Hash (REwH1) scheme [6] builds on a PKE schem $\mathcal{AE}_r := (\mathsf{KGen}_r, \mathcal{E}_r, \mathcal{D}_r)$ in the random oracle model. The REwH1 scheme inherits key generation KGen and decryption $\mathcal{D}$ from $\mathcal{AE}_r$, while encryption is defined as

$$\mathcal{E}^{\mathcal{R}}(pk, m; r) := \mathcal{E}_r\left(pk, m; \mathcal{R}(pk \| m \| r)\right) \ .$$

F.1 Adaptive IK-CPA

For our result we need to adapt the IK-CPA notion such that the adversary can adaptively generate ciphertexts. Let us call the adaptive notion aIK-CPA. We depict the corresponding security game in Figure 22. As is the case for the standard IND-CPA notion for public key encryption, IK-CPA implies aIK-CPA. The proof follows from a standard hybrid argument.

Proposition F.1. *Let $\mathcal{A}$ be an aIK-CPA adversary making at most t queries to oracle* `LoR`*. Then there exists adversary $\mathcal{B}$ running in time of $\mathcal{A}$, such that*

$$\mathbf{Adv}^{\mathit{aIK\text{-}CPA}}_{\mathcal{AE}}(\mathcal{A}) \leq t \cdot \mathbf{Adv}^{\mathit{IK\text{-}CPA}}_{\mathcal{AE}}(\mathcal{B})$$

We define $\text{IK-CPA0}^{\mathcal{A}_1,\mathcal{A}_2}_{\mathcal{AE}}$ exactly as $\text{IK-CPA}^{\mathcal{A}_1,\mathcal{A}_2}_{\mathcal{AE}}$ except that bit b is set to zero at the beginning of the game and the game returns the guess of adversary $\mathcal{A}_2$. Likewise, we define $\text{IK-CPA1}^{\mathcal{A}_1,\mathcal{A}_2}_{\mathcal{AE}}$ where bit b is set to one. This allows us to write the advantage of an adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$ against IK-CPA as:

$$\mathbf{Adv}^{\text{IK-CPA}}_{\mathcal{AE}}(\mathcal{A}) := \Pr\left[\,\text{IK-CPA1}^{\mathcal{A}_1,\mathcal{A}_2}_{\mathcal{AE}} \Rightarrow 1\,\right] - \Pr\left[\,\text{IK-CPA0}^{\mathcal{A}_1,\mathcal{A}_2}_{\mathcal{AE}} \Rightarrow 1\,\right]$$

Similarly, we define games for the adaptive version:

$$\mathbf{Adv}^{\text{aIK-CPA}}_{\mathcal{AE}}(\mathcal{A}) := \Pr\left[\,\text{aIK-CPA1}^{\mathcal{A}}_{\mathcal{AE}} \Rightarrow 1\,\right] - \Pr\left[\,\text{aIK-CPA0}^{\mathcal{A}}_{\mathcal{AE}} \Rightarrow 1\,\right] \tag{4}$$

Proof of Proposition F.1. Let, without loss of generality, adversary $\mathcal{A}$ make exactly t queries. We define a sequence of adversaries $\mathcal{B}^i := (\mathcal{B}^i_1, \mathcal{B}^i_2)$ (for $0 < i \leq t$) against IK-CPA having access to an adversary $\mathcal{A}$ against aIK-CPA. Adversary $\mathcal{B}^i_1$ gets as input two public keys pk_0, pk_1. It simulates oracle `LoR` as follows: for the first $i-1$ queries m it answers with $\mathcal{E}(pk_0, m)$. For the i-th query m, adversary $\mathcal{B}^i_1$ simply

game aIK-CPA$^{\mathcal{A}}_{\mathcal{AE}}(1^\lambda)$	**procedure** LoR(m)
$b \leftarrow \{0,1\}$	return $\mathcal{E}(pk_b, m)$
$(pk_0, sk_0) \leftarrow \mathsf{KGen}(1^\lambda)$	
$(pk_1, sk_1) \leftarrow \mathsf{KGen}(1^\lambda)$	
$b' \leftarrow \mathcal{A}^{\text{LoR}}(pk_1, pk_2)$	
return $(b = b')$	

Figure 22: Adaptive Key Indistinguishability under Chosen-Plaintext Attack

outputs m together with its state. Adversary $\mathcal{B}_2^i$ receives as input ciphertext c which is either $\mathcal{E}(pk_0, m)$ or $\mathcal{E}(pk_1, m)$ and the state. It returns c as answer to the LoR-query. Adversary $\mathcal{B}_2^i$ answers all further LoR-queries m from adversary $\mathcal{A}$ with $\mathcal{E}(pk_1, m)$ and outputs as guess for bit b whatever adversary $\mathcal{A}$ outputs.

If b equals zero, then adversary $\mathcal{B}^t$ perfectly simulates the LoR oracle and likewise if b equals 1 then adversary $\mathcal{B}^0$ perfectly simulates the LoR oracle. Let $\mathcal{B} := (\mathcal{B}_1, \mathcal{B}_2)$ be the adversary that chooses $0 < i \leq t$ uniformly at random to then implement adversary $\mathcal{B}^i$. Thus, we have that

$$\begin{aligned}\Pr[\,\mathcal{B} \text{ outputs } 0 | b = 0\,] &:= \sum_{j=1}^{t} \Pr\left[\,\mathcal{B}^j \text{ outputs } 0 | b = 0 \wedge i = j\,\right] \cdot \Pr[\,i = j\,] \\ &= \frac{1}{t} \sum_{j=1}^{t} \Pr\left[\,\mathcal{A}^j \text{ outputs } 0\,\right] \end{aligned} \tag{5}$$

where $\mathcal{A}^j$ is the adversary in the adaptive aIK-CPA game getting an LoR-oracle that on the first j queries uses public key pk_0 and on the remaining queries uses public key pk_1. Likewise, we have that

$$\begin{aligned}\Pr[\,\mathcal{B} \text{ outputs } 1 | b = 1\,] &:= \sum_{j=1}^{t} \Pr\left[\,\mathcal{B}^j \text{ outputs } 1 | b = 1 \wedge i = j\,\right] \cdot \Pr[\,i = j\,] \\ &= \frac{1}{t} \sum_{j=0}^{t-1} \Pr\left[\,\mathcal{A}^j \text{ outputs } 1\,\right] \end{aligned} \tag{6}$$

Putting it all together, we have that

$$\begin{aligned}\mathbf{Adv}^{\text{IK-CPA}}_{\mathcal{AE}}(\mathcal{B}) &= \Pr\left[\,\text{IK-CPA1}^{\mathcal{B}_1,\mathcal{B}_2}_{\mathcal{AE}} \Rightarrow 1\,\right] - \Pr\left[\,\text{IK-CPA0}^{\mathcal{B}_1,\mathcal{B}_2}_{\mathcal{AE}} \Rightarrow 1\,\right] \\ &= \Pr\left[\,\text{IK-CPA1}^{\mathcal{B}_1,\mathcal{B}_2}_{\mathcal{AE}} \Rightarrow 1\,\right] - 1 + \Pr\left[\,\text{IK-CPA0}^{\mathcal{B}_1,\mathcal{B}_2}_{\mathcal{AE}} \Rightarrow 0\,\right]\end{aligned}$$

With equations (5) and (6) this yields

$$\begin{aligned}&= \Pr[\,\mathcal{B} \text{ outputs } 1 | b = 1\,] + \Pr[\,\mathcal{B} \text{ outputs } 0 | b = 0\,] - 1 \\ &= \frac{1}{t}\left(\sum_{j=0}^{t-1} \Pr\left[\,\mathcal{A}^j \text{ outputs } 1\,\right] + \sum_{j=1}^{t} \Pr\left[\,\mathcal{A}^j \text{ outputs } 0\,\right]\right) - 1 \\ &= \frac{1}{t}\left(\Pr\left[\,\mathcal{A}^0 \text{ outputs } 1\,\right] + \Pr\left[\,\mathcal{A}^t \text{ outputs } 0\,\right]\right) + \\ &\quad \frac{1}{t}\sum_{j=1}^{t-1}\left(\Pr\left[\,\mathcal{A}^j \text{ outputs } 1\,\right] + \Pr\left[\,\mathcal{A}^j \text{ outputs } 0\,\right]\right) - 1\end{aligned}$$

As $\left(\Pr\left[\,\mathcal{A}^j \text{ outputs } 1\,\right] + \Pr\left[\,\mathcal{A}^j \text{ outputs } 0\,\right]\right) = 1$ for all $1 \leq j \leq t-1$ this is

$$\begin{aligned}&= \frac{1}{t}\left(\Pr[\,\text{aIK-CPA1} \Rightarrow 1\,] + \Pr[\,\text{aIK-CPA0} \Rightarrow 0\,]\right) - \frac{1}{t} \\ &= \frac{1}{t}\left(\Pr[\,\text{aIK-CPA1} \Rightarrow 1\,] - \Pr[\,\text{aIK-CPA0} \Rightarrow 1\,] + 1\right) - \frac{1}{t}\end{aligned}$$

Finally, with equation (4) we get the advantage statement of the theorem:

$$= \frac{1}{t}\mathbf{Adv}^{\text{aIK-CPA}}_{\mathcal{AE}}(\mathcal{A})$$

which concludes the proof. □

F.2 REwH1 with IK-CPA implies PK-EXT

We can now show that the Randomized-Encrypt-with-Hash scheme is PK-EXT-secure if the underlying PKE scheme is IK-CPA-secure. We only consider the PK-EXT-notion in the random oracle model. Note that, as it is a single-stage notion this suffices for composition in the MRH theorem. Remember that $\mathsf{maxpk}_{\mathcal{AE}}$ denotes the maximum probability of a collision for a public-key as generated by KGen, defined in equation (3).

Theorem F.2. *Let $\mathcal{A}$ be a PK-EXT adversary making at most $q_{\mathcal{A}}$ random oracle queries. Then there exists an adversary $\mathcal{B}$ running in time of $\mathcal{A}$, such that*

$$\mathbf{Adv}^{PK\text{-}EXT}_{REwH1}(\mathcal{A}) \leq \mathbf{Adv}^{aIK\text{-}CPA}_{\mathcal{AE}}(\mathcal{B}) + (q_{\mathcal{A}} + 1) \cdot \mathsf{maxpk}_{\mathcal{AE}} \ .$$

Proof. We assume, without loss of generality, that adversary $\mathcal{A}$ does not repeat queries to its oracles.

We define adversary $\mathcal{B}$ against aIK-CPA. Adversary $\mathcal{B}$ gets as input two public keys pk_0 and pk_1 and runs adversary $\mathcal{A}$ against PK-EXT. It simulates $\mathcal{A}$'s queries to the $\texttt{ENC}$ oracle using its $\texttt{LoR}$-oracle simply ignoring the randomness. That is, if (m, r) is a query by $\mathcal{A}$ to the $\texttt{ENC}$-oracle, then $\mathcal{B}$ answers this as $\texttt{LoR}(m)$. Let q be a query to the random oracle by $\mathcal{A}$. Before answering, $\mathcal{B}$ tests if the first bits of query q equal one of the public keys, that is, if

$$q_{|_{1,\ldots,|pk_0|}} = pk_0 \qquad \text{or if} \qquad q_{|_{1,\ldots,|pk_1|}} = pk_1 \ .$$

If this is the case, then $\mathcal{B}$ terminates $\mathcal{A}$ and outputs 0 if the first bits were equal to pk_0 and 1 otherwise. If the first bits did not equal either of the keys it responds with $\mathcal{R}(q)$. If adversary $\mathcal{A}$ terminates with guess pk' then adversary $\mathcal{B}$ outputs 0 if $pk' = pk_0$, it outputs 1 if $pk' = pk_1$, and else outputs a random bit.

Let the event $\mathsf{bad1}$ be defined as $\mathcal{A}$ queries its random oracle on message $pk_{1-b}\|x$ where x is a some bit string, which in turn leads to $\mathcal{B}$ outputting a wrong guess for b. As no information about pk_{1-b} is leaked to adversary $\mathcal{A}$ we can bind the probability of $\mathsf{bad1}$ via a union bound with

$$\Pr[\,\mathsf{bad1}\,] \leq q_{\mathcal{A}} \cdot \mathsf{maxpk}_{\mathcal{AE}}$$

where $q_{\mathcal{A}}$ denotes the number of random oracle queries by adversary $\mathcal{A}$.

Let the event $\mathsf{bad2}$ be defined as $\mathcal{A}$ outputs guess $pk' = pk_{1-b}$. With the same argument this probability is bound by

$$\Pr[\,\mathsf{bad2}\,] \leq \mathsf{maxpk}_{\mathcal{AE}}$$

If events $\mathsf{bad1}$ and $\mathsf{bad2}$ do not occur then note that adversary $\mathcal{B}$ perfectly simulates the oracles that are expected by $\mathcal{A}$. Adversary $\mathcal{A}$ expects the encryption scheme to use randomness generated as $\mathcal{R}(pk\|m\|r)$. This means that as $\mathcal{A}$ never queries the random oracle on $pk\|m\|r$ (this is implied by $\neg\mathsf{bad1}$), it must expect the scheme to use uniformly random coins. This is exactly what is done by the $\texttt{LoR}$ oracle. In this case adversary $\mathcal{B}$ wins whenever $\mathcal{A}$ outputs a correct guess (or queries the random oracle on $pk_b\|x$). This concludes the proof. □

A simple corollary of the theorem is, that if the adversary in the PK-EXT game is not allowed to specify the randomness used by the encryption scheme, then PK-EXT is directly implied by IK-CPA.

Corollary F.3. *Let $\mathcal{A}$ be a PK-EXT adversary which is not allowed to specify the randomness used by the encryption scheme. Then there exists an adversary $\mathcal{B}$ running in time of $\mathcal{A}$, such that*

$$\mathbf{Adv}^{PK\text{-}EXT}_{\mathcal{AE}}(\mathcal{A}) \leq \mathbf{Adv}^{aIK\text{-}CPA}_{\mathcal{AE}}(\mathcal{B}) + (q_{\mathcal{A}} + 1) \cdot \mathsf{maxpk}_{\mathcal{AE}} \ .$$