On the Indifferentiability of the Grøstl Hash Function

Elena Andreeva, Bart Mennink and Bart Preneel

Dept. Electrical Engineering, ESAT/COSIC
Katholieke Universiteit Leuven, Belgium
{elena.andreeva, bart.mennink, bart.preneel}@esat.kuleuven.be

Abstract. The notion of indifferentiability, introduced by Maurer et al., is an important criterion for the security of hash functions. Concretely, it ensures that a hash function has no structural design flaws and thus guarantees security against generic attacks up to the exhibited bounds. In this work we prove the indifferentiability of Grøstl, a second round SHA-3 hash function candidate. Grøstl combines characteristics of the wide-pipe and chop-Merkle-Damgård iterations and uses two distinct permutations P and Q internally. Under the assumption that P and Q are random l-bit permutations, where l is the iterated state size of Grøstl, we prove that the advantage of a distinguisher to differentiate Grøstl from a random oracle is upper bounded by $O((Kq)^4/2^l)$, where the distinguisher makes at most q queries of length at most K blocks. For the specific Grøstl parameters, this result implies that Grøstl behaves like a random oracle up to $q = O(2^{n/2})$ queries, where n is the output size.
Furthermore, we show that the output transformation of Grøstl, as well as 'Grøstail' (the composition of the final compression function and the output transformation), are clearly differentiable from a random oracle. This renders out indifferentiability proofs which rely on the idealness of a final state transformation.

1 Introduction

Hash functions are a basic building block in cryptography. Formally, a hash function maps a bit string of arbitrary length to an output string of fixed length, $\mathcal{H} : \mathbb{Z}_2^* \to \mathbb{Z}_2^l$. An established practice in the design of hash functions is to first construct a fixed input length compression function, e.g. $f : \mathbb{Z}_2^l \times \mathbb{Z}_2^l \to \mathbb{Z}_2^l$, and then iterate it to allow the processing of arbitrarily long strings. The most popular iteration principle is the strengthened Merkle-Damgård [13, 21] design[1]. Common hash functions, such as members of the SHA and MD family, incorporate the Merkle-Damgård method in their design. However, recent attacks on the widely used SHA-1 and MD5 [25, 26] have rendered these designs insecure. This grim situation has triggered the launch of the SHA-3 competition [23] for the selection of a new secure hash function algorithm by NIST (National Institute of Standards and Technology). In the current second round of the competition, 14 candidates are under active evaluation.
These 14 candidates use a wide variety of iterative modes. Some of the designs still follow the basic Merkle-Damgård iteration. Others either add new features to it, or simply propose different constructions. Candidates from the latter two classes include iterations based on the chop-Merkle-Damgård [14], HAIFA [7], wide-pipe [19] and Sponge [5] design strategies. The main advantage of the basic Merkle-Damgård construction is its collision security guarantee under the assumption that the underlying compression function is collision resistant [13, 21]. Other important hash function security properties, such as second preimage and preimage security are, however, not preserved by the Merkle-Damgård iteration [1]. Moreover, the

[1] Throughout, we will refer to it as the 'Merkle-Damgård design'.

extension attack shows that the Merkle-Damgård hash function is clearly differentiable from a monolithic random oracle [12].
A natural question that arises with the emerge of new iterative designs is to identify the security properties achieved by these constructions. Other than the classical collision, second preimage and preimage security properties, the *indifferentiability* property has gained more recent attention due to the advancements in the theoretical differentiability model of Maurer et al. [20] and their further development in the context of hashing [12, 2, 10, 18, 11, 15]. Indifferentiability is an important security criterion because it ensures that the hash function has no structural design flaws in composition. Such a result provides a guarantee that no generic attacks (attacks on the iteration which assume ideal behavior of the underlying primitives) up to the proven bounds are possible.
In this work we analyze the indifferentiability of the Grøstl SHA-3 candidate [17]. Grøstl borrows characteristics mainly from the wide-pipe and the chop-Merkle-Damgård iterations: the iterated state is wider than the final hash output, which classifies it as a type of a wide-pipe design. The iterative message processing together with a final state truncation in Grøstl resemble the chop-Merkle-Damgård hash function with the added difference of a present output transformation before truncation. More concretely, Grøstl processes its inputs by first calling the compression function f iteratively, then applying a final output transformation to the state and finally truncating the result to the desired output length. The compression function f is built out of two permutations P and Q and the output transformation is designed on top of the permutation P.

1.1 Our Result

Indifferentiability results on hash functions can be obtained following several different approaches. One way to argue indifferentiability is to assume ideal behavior of the first layer components (i.e., the underlying compression functions), and prove the result for the concrete composition of interest [12, 11]. Dodis et al. [16] relax the assumption on the internal compression functions from a random oracle to preimage awareness. If a composition is preimage aware, which they show is true for the Merkle-Damgård iteration when the compression function is preimage aware itself, then they prove indifferentiability by assuming idealness only of the final extra transformation. Both approaches turn out futile for the Grøstl hash function: fixed points for the compression function can be found easily (as already observed in [17]), and also the final output transformation is clearly differentiable from a random function. Even stronger, if we consider the composition of the final compression function f and the output transformation (with and without truncation), which we refer to as 'Grøstail', then we prove that Grøstail is differentiable from a random function. We do so by demonstrating an attack that tricks any simulator for the indifferentiability of Grøstail in only three oracle queries. This result indicates that Grøstail is highly non-random and therefore the results of [16] could not be applied directly.
The next attempt for an indifferentiability proof for the Grøstl hash function is to refine the level of modularity and to explore the second layer integral components, i.e. the permutations P and Q. In a similar fashion, Coron et al. [12] prove that the chop-Merkle-Damgård construction with Davies-Meyer (DM) [24] compression function is indifferentiable from a random oracle assuming an ideal behavior from the block cipher underlying the DM function. While the Grøstl iteration is a type of a DM chop-Merkle-Damgård construction,

the latter result cannot be applied here due to clear design differences, such as the presence of an output transformation. Instead, to prove indifferentiability of the Grøstl hash function we start from scratch by assuming ideal behavior of the underlying permutations.
The proof is constructed following the indifferentiability theoretical framework by [20]. We build a simulator for the permutations P and Q that is granted access to a random oracle. The goal of the simulator is to answer its queries, such that it is hard for a distinguisher to tell apart the interactions with the Grøstl hash functions and truly random permutations from the ones with a random oracle and the simulator. The simulator is also consistent with the outputs of the random oracle.
We prove that the advantage of a distinguisher to differentiate Grøstl from a random oracle is upper bounded by $O((Kq)^4/2^l)$, where the distinguisher makes at most q queries of length at most K blocks to its oracles. Here, l is the iterated state size and n the output hash size of Grøstl: $l = 512$ for $n \leq 256$ and $l = 1024$ for $n \leq 512$. Intuitively, this means that Grøstl behaves like a random oracle up to $q = O(2^{n/2})$ queries.
The JH [6], Keccak [4] and Shabal [8, 9] SHA-3 second round candidates have recently been also proved indifferentiable. All of them claim security beyond the birthday bound (with respect to the output length n). In particular, JH is proven indifferentiable up to $O(q^3/2^{l-m})$, and Keccak and Shabal up to $O((Kq)^2/2^{l-m})$ where l is the size of the chaining value and m the number of message bits compressed in one application of the compression function. We notice however, that this is an unfair comparison: JH, Keccak and Shabal have iterated state sizes l of $1024, 1600$ and 1408 bits, respectively, which are significantly larger than the state size of Grøstl. For comparison, Keccak-256 is indifferentiable up to bound $O((Kq)^2/2^{512})$, while our result implies that Grøstl-256 would be indifferentiable up to $O((Kq)^4/2^{1600})$, were Grøstl be designed to have the same state size as Keccak.

2 Preliminaries

For $n \in \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers, let $\mathbb{Z}_2^n$ denote the set of bit strings of length n, $(\mathbb{Z}_2^n)^*$ the set of strings of length a multiple of n and $\mathbb{Z}_2^*$ the set of strings of arbitrary length. If x, y are strings, then $x\|y$ is the concatenation of x and y. If $k, l \in \mathbb{N}$ then $\langle k \rangle_l$ is the encoding of k as an l-bit string. If S is a set, then $x \xleftarrow{\$} S$ denotes the uniformly random selection of an element from S. We let $y \leftarrow \mathsf{A}(x)$ and $y \xleftarrow{\$} \mathsf{A}(x)$ be the assignment to y of the output of a deterministic and randomized algorithm A, respectively, when run on input x. For a function f, by $\mathrm{dom}(f)$ and $\mathrm{rng}(f)$ we denote the domain and range of f, respectively. Abusing notation, by $(x, y) \in f$, we denote that $x \in \mathrm{dom}(f)$ and $y = f(x)$.
A random oracle [3] is a function which provides a random output for each new query. A random l-bit permutation is a function that is taken uniformly at random from the set of all l-bit permutations. A random primitive will also be called 'ideal'.

2.1 Grøstl

On input of a message of arbitrary length, the Grøstl hash function $\mathsf{Gr} : \mathbb{Z}_2^* \to \mathbb{Z}_2^n$ outputs a digest of n bits, with $n \in \{224, 256, 384, 512\}$ [17]. Grøstl is a type of a wide-pipe design where the iterated state size l is significantly larger than the final hash output. More concretely: for $n = 224, 256$, $l = 512$, and for $n = 384, 512$, $l = 1024$. The Grøstl hash function

makes use of the Merkle-Damgård construction to process its inputs, then applies an output transformation on the state value and finally truncates (chops) the result from l to n bits. The Grøstl compression function $f : \mathbb{Z}_2^l \times \mathbb{Z}_2^l \to \mathbb{Z}_2^l$ is defined as $f(h, m) = P(h \oplus m) \oplus Q(m) \oplus h$, where $P, Q : \mathbb{Z}_2^l \to \mathbb{Z}_2^l$ are two l-bit permutations. Throughout, P and Q are considered to be independent random permutations.

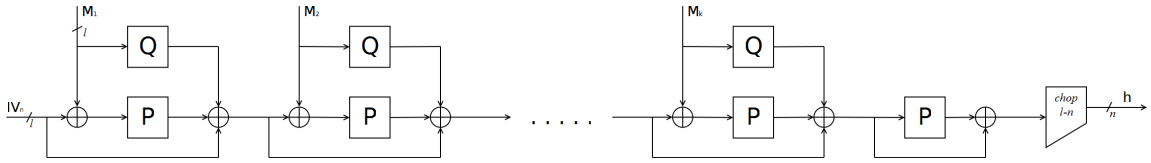

Fig. 1. The Grøstl hash function Gr.

For a fixed initialization vector IV_n the hash function Gr (see Fig. 1) processes an arbitrary length message M as follows:

$$\begin{aligned}(M_1, \ldots, M_k) &= \mathrm{pad}(M),\\ h_0 &= IV_n,\\ h_i &= f(h_{i-1}, M_i) \text{ for } i = 1, \ldots, k,\\ h_{k+1} &= P(h_k) \oplus h_k,\\ \mathsf{Gr}(M) &= \mathrm{chop}_{l-n}(h_{k+1}),\end{aligned}$$

where $\mathrm{chop}_{l-n}(x)$ chops off the $l - n$ rightmost bits of x, and the padding function pad is defined as $\mathrm{pad}(M) = M'$, with $M' = M \| 1 \| 0^{-|M|-65 \bmod l} \| \langle \lceil (|M| + 65)/l \rceil \rangle_{64}$, parsed as a sequence of l-bit blocks. On input of a message $M' \in \left(\mathbb{Z}_2^l\right)^*$, the function $\mathrm{depad}(M')$ is defined as follows: if $M' = \mathrm{pad}(M)$ for some message M, it outputs this M, otherwise it outputs $\perp$. Observe that the output is unique as the padding function is injective. For an $M \in \left(\mathbb{Z}_2^l\right)^*$, we denote by $Z(M)$ the set of all values $m \in \mathbb{Z}_2^l$ that make (M, m) a valid padding. Formally: $Z(M) = \{m \in \mathbb{Z}_2^l \mid \mathrm{depad}(M, m) \neq \perp\}$. Apart from the indifferentiability of the Grøstl hash function, we also consider the Grøstail function $\mathsf{F} : \mathbb{Z}_2^l \times \mathbb{Z}_2^l \to \mathbb{Z}_2^l$, a composition of the last compression function f with the final transformation (i.e., Grøstail is the 'tail' of Grøstl):

$$\mathsf{F}(h, m) = P(f(h, m)) \oplus f(h, m). \tag{1}$$

2.2 Indifferentiability

The indifferentiability framework introduced by Maurer et al. [20] is an extension of the classical notion of indistinguishability. It proves that if a construction $\mathcal{C}^{\mathcal{G}}$ based on an ideal subcomponent $\mathcal{G}$ is indifferentiable from an ideal primitive $\mathcal{R}$, then $\mathcal{C}^{\mathcal{G}}$ can replace $\mathcal{R}$ in any system.

Definition 1. *A Turing machine $\mathcal{C}$ with oracle access to an ideal primitive $\mathcal{G}$ is said to be $(t_D, t_S, q, \varepsilon)$ indifferentiable from an ideal primitive $\mathcal{R}$ if there exists a simulator $\mathcal{S}$, such that for any distinguisher $\mathcal{D}$ it holds that:*

$$\mathrm{Adv}^{\mathrm{pro}}_{\mathcal{C},\mathcal{S}}(\mathcal{D}) = \left|\Pr\left(\mathcal{D}^{\mathcal{C}^{\mathcal{G}},\mathcal{G}} = 1\right) - \Pr\left(\mathcal{D}^{\mathcal{R},\mathcal{S}^{\mathcal{R}}} = 1\right)\right| < \varepsilon.$$

The simulator has oracle access to $\mathcal{R}$ and runs in time at most t_S. The distinguisher runs in time at most t_D and makes at most q queries.

In the remainder, we refer to $\mathcal{C}^{\mathcal{G}}, \mathcal{G}$ as the 'real world', and to $\mathcal{R}, \mathcal{S}^{\mathcal{R}}$ as the 'simulated world'; the distinguisher $\mathcal{D}$ converses either with the real or the simulated world and its goal is to tell both worlds apart. $\mathcal{D}$ can query both its 'left oracle' L (either $\mathcal{C}$ or $\mathcal{R}$) and its 'right oracle' R (either $\mathcal{G}$ or $\mathcal{S}$). In the remainder, R has four interfaces, corresponding to forward and inverse queries to permutations P and Q. These interfaces are denoted by $R_P, R_{P^{-1}}, R_Q, R_{Q^{-1}}$.

3 Differentiability of Grøstail

A recent result by Dodis et al. [16] prescribes how to prove indifferentiability of hash functions by ways of preimage awareness. Loosely speaking, Dodis et al. proved that if $\mathcal{H} : \mathbb{Z}_2^* \to \mathbb{Z}_2^l$ is a preimage aware hash function and $RO : \mathbb{Z}_2^l \to \mathbb{Z}_2^l$ is a random function, then the composition $RO \circ \mathcal{H}$ is indifferentiable from a random oracle. One might be tempted to consider this approach for the indifferentiability analysis of Grøstl, i.e., by assuming that the output transformation is a random oracle and then prove the Grøstl hash function (without the output transformation) to be preimage aware. However, the behavior of the output transformation $P(x) \oplus x$ deviates significantly from a random function: similarly to the Davies-Meyer construction [22], fixed points $P(x) \oplus x = x$ are easy to compute by making the inverse query $P^{-1}(0) = x$. A second attempt is to go one step backwards in the iteration and view the last compression function together with the output transformation, i.e., Grøstail (1), as a random function. We show that this approach also fails since Grøstail is easily differentiable from a random function.

Proposition 1. *Let P, Q be two random l-bit permutations, let F be the Grøstail compression function (1), and let $RO : \mathbb{Z}_2^l \times \mathbb{Z}_2^l \to \mathbb{Z}_2^l$ be a random function. For any simulator S that makes at most q queries to RO, there exists a distinguisher $\mathcal{D}$ that makes at most 3 queries to its oracle, such that:*

$$\mathrm{Adv}^{\mathrm{pro}}_{\mathsf{F},\mathsf{S}}(\mathcal{D}) \geq 1 - \frac{q}{2^l}.$$

Proof. Let S be any simulator that makes at most q queries to RO. We construct a distinguisher $\mathcal{D}$ that with overwhelming probability distinguishes Grøstail from a random function in 3 oracle queries. The distinguisher proceeds as follows. First, it makes inverse queries $x_2 = R_{Q^{-1}}(0)$ and $x_1 = R_{P^{-1}}(x_2)$. Then, it makes a query to the left oracle to obtain $y = L(x_1 \oplus x_2, x_2)$. If $\mathcal{D}$ converses with $(\mathsf{F}^{P,Q}, (P, Q))$, then, by construction:

$$y = \mathsf{F}^{P,Q}(x_1 \oplus x_2, x_2) = P(x_1) \oplus x_1 = x_1 \oplus x_2.$$

If $\mathcal{D}$ converses with (RO, S^{RO}), this equation holds only if the simulator can find x_1, x_2 such that $RO(x_1 \oplus x_2, x_2) = x_1 \oplus x_2$, i.e., only if the simulator can find a fixed point for RO. As the probability for the simulator to find fixed points for RO is upper bounded by $q/2^l$, the advantage for $\mathcal{D}$ to distinguish, $\mathrm{Adv}^{\mathrm{pro}}_{\mathsf{F},\mathsf{S}}(\mathcal{D})$, is lower bounded by $1 - q/2^l$. □

If the final truncation is included in Grøstail as well, a lower bound $1 - q/2^n$ can be obtained similarly.

4 Indifferentiability of Grøstl

In this section, we present the main result of this paper: we show that the Grøstl hash function is indifferentiable from a random oracle, under the assumption that the underlying permutations P, Q are ideal. Intuitively, we demonstrate that there exists a simulator such that no distinguisher can differentiate the real world $\mathsf{Gr}^{P,Q}, (P, Q)$ from the simulated world RO, S^{RO}, except with negligible probability.

Theorem 1. *Let P, Q be two random l-bit permutations, let Gr be the Grøstl hash function (Sect. 2.1), and let RO be a random oracle. Let $\mathcal{D}$ be a distinguisher that makes at most q_L left queries of maximal length $(K-1)l$ bits, where $K \geq 1$, q_P right queries to P and q_Q right queries to Q, and runs in time t. Then:*

$$\mathrm{Adv}^{\mathrm{pro}}_{\mathsf{Gr},\mathsf{S}}(\mathcal{D}) \leq \frac{58(q_P + (K+1)q_L)^2(q_Q + Kq_L)^2}{2^l}, \tag{2}$$

where S makes at most $q_S \leq q_P$ queries to RO and runs in time $O(\max\{q_P, q_Q\}^4)$.

The simulator S used in the proof mimics the behavior of random permutations P and Q such that queries to S and queries to RO are 'consistent', which means that relations among the query outputs in the real world hold in the simulated world as well. To this end, the construction of the simulator is based on several designing decisions. In what remains, the simulator used in the proof (Fig. 2) is introduced and explained in more detail. Then, Thm. 1 is proven in Sect. 4.3.

4.1 Initialization of the Simulator

The simulator maintains two, initially empty, databases P, Q that represent the permutations it simulates. It has four interfaces, denoted by $\mathsf{S}_P, \mathsf{S}_{P^{-1}}, \mathsf{S}_Q, \mathsf{S}_{Q^{-1}}$, and access to RO. Furthermore, the simulator maintains a graph (V, E), initially $(\{IV\}, \emptyset)$. The edges $e \in E$ are labeled by messages in $\mathbb{Z}_2^l$: any $(x_1, y_1) \in P$ and $(x_2, y_2) \in Q$ define an edge $x_1 \oplus x_2 \xrightarrow{x_2} x_1 \oplus x_2 \oplus y_1 \oplus y_2$ in (V, E). Intuitively, an edge in (V, E) corresponds to an evaluation of the Grøstl compression function f, and if there is a path $IV \xrightarrow{M_1} s_1 \xrightarrow{M_2} \cdots \xrightarrow{M_k} s_k$ in (V, E), then $f(\ldots f(f(IV, M_1), M_2) \ldots, M_k) = s_k$. Abusing notation, we denote by $s \xrightarrow{M} t$ that there is a path from s to t in (V, E) with the edges labeled by $M \in \left(\mathbb{Z}_2^l\right)^*$. We say that (V, E) contains *colliding paths* if there exists an $s \in V$ such that $IV \xrightarrow{M} s$ and $IV \xrightarrow{M'} s$ are two paths in (V, E), for different $M, M' \in \left(\mathbb{Z}_2^l\right)^*$.
Furthermore, by $V_{\mathrm{out}}, V_{\mathrm{in}}$ we denote the set of vertices in V with an outgoing or ingoing edge, respectively. Observe that after q_P, q_Q queries to P, Q, the sets $V_{\mathrm{out}}, V_{\mathrm{in}}$ are of size at most $q_P q_Q$. By $r(V)$, we denote the set of all *rooted* nodes in V, i.e.: $r(V) = \left\{ v \in V \;\middle|\; \exists\, M \in \left(\mathbb{Z}_2^l\right)^* \text{ such that } IV \xrightarrow{M} v \right\}$. By construction, $r(V) \subseteq V_{\mathrm{in}}$. Finally, we introduce a specific subset of $r(V)$:

$$\bar{r}(V) = \left\{ v \in V \;\middle|\; \exists\, M \in \left(\mathbb{Z}_2^l\right)^* \text{ such that } IV \xrightarrow{M} v \text{ and } \mathrm{depad}(M) \neq \perp \right\}.$$

For simplicity, $V, r(V)$ and $\bar{r}(V)$ are updated by the simulator implicitly.

4.2 Intuition Behind the Simulator

In this section we take a closer look at the simulator of Fig. 2 by starting with an example. Consider the case that a node x is a member of both $\bar{r}(V)$ and $\mathrm{dom}(P)$. This means that (1) there exists an M such that $IV \xrightarrow{M} x$ and $\mathrm{depad}(M) \neq \perp$, and (2) there exists a $y \in \mathrm{rng}(P)$, such that $y = P(x)$. In the real world (where the left oracle is the Grøstl hash function), these values satisfy $\mathsf{Gr}(\mathrm{depad}(M)) = \mathrm{chop}_{l-n}(x \oplus y)$ by construction. If the simulator does not answer its queries wisely, this equality would hold with negligible probability in the simulated world. More generally, the simulator can guarantee that this equation holds only if x is added to $\mathrm{dom}(P)$ *after* it was added to $\bar{r}(V)$ (reflected in requirement R3 below)[2]. Maintaining consistency, however, becomes harder when $|\bar{r}(V)|$ and $|\mathrm{dom}(P)|$ increase. The idea behind the simulator is to answer its queries such that it can control the growth of $r(V)$, and in particular the growth of $\bar{r}(V)$ as a subset of $r(V)$, while still maintaining consistency in its answers. Intuitively, the simulator responds to its queries, such that the following requirements are satisfied:

R1. There are no colliding paths in $(r(V), E)$. Observe that two different paths to the same node may lead to distinguishability for $\mathcal{D}$ as the simulator can be consistent with only *one* of the paths. This requirement is satisfied if $r(V)$ is never increased with a node that has two incoming edges in the updated[3] graph;
R2. S increases $r(V)$ only if it is forced to do. In particular, $r(V)$ is never increased with a node that has an outgoing edge in the updated graph. Observe that each path in $(r(V), E)$ leads to a potential node in $\bar{r}(V)$;
R3. S never increases $\bar{r}(V)$ with a node that is a member of the updated $\mathrm{dom}(P)$;
R4. S increases $\mathrm{dom}(P)$ with a node in $\bar{r}(V)$ only if it is forced to. Observe that in case of inverse queries to P, the simulator can avoid outputting elements in $\bar{r}(V)$. In forward queries to P, the simulator may be forced to increase $\bar{r}(V) \cap \mathrm{dom}(P)$. In this case, it consults its oracle RO to generate the answer.

The first two conditions are regarding the growth of $r(V)$, and the second two concern the growth of $\bar{r}(V) \cap \mathrm{dom}(P)$. We show how these conditions occur in the description of the simulator in Fig. 2. We first consider requirements R1 and R2, then we look at R3 and R4.

Restricting the growth of $r(V)$

Inverse queries. Consider an inverse query y_1 to $\mathsf{S}_{P^{-1}}$. It is easy to see that both R1 and R2 are satisfied if the simulator outputs its answer x_1, such that none of the newly added vertices $\{x_1 \oplus x_2 \mid x_2 \in \mathrm{dom}(Q)\}$ to V_{out} is already rooted. A similar observation holds for queries to $\mathsf{S}_{Q^{-1}}$. These requirements translate to lines 3e and 4c in the description of the simulator in Fig. 2.

---

[2] Observe that $RO(\mathrm{depad}(M)) = \mathrm{chop}_{l-n}(\mathsf{S}_P(x) \oplus x)$ should hold for $IV \xrightarrow{M} x$. If $x \in \mathrm{dom}(P)$ before it is added to $\bar{r}(V)$, this means that $\mathsf{S}_P(h_k) \oplus x$ is fixed before $RO(\mathrm{depad}(M))$ is known.

[3] This requirement should hold for the 'updated' graph, which can be seen as follows: suppose the distinguisher makes a forward query x_1 to S_P such that $x_1 \oplus x_2, x_1 \oplus x_2' \in r(V)$ for different $x_2, x_2' \in \mathrm{dom}(Q)$, and both $x_1 \oplus x_2 \oplus y_1 \oplus y_2$ and $x_1 \oplus x_2' \oplus y_1 \oplus y_2'$ are not in V yet. By construction, these nodes have zero incoming edges in the non-updated (V, E), but it may accidentally be the case that these nodes are equal, in which case they have two incoming edges in the updated graph.

FORWARD QUERIES. In forward queries to $\mathsf{S}_P, \mathsf{S}_Q$, the simulator may be forced to increase $r(V)$. Consider a query x_1 to S_P, and consider any $x_2 \in \mathrm{dom}(Q)$ such that $x_1 \oplus x_2 \in r(V)$. Then, the edge $x_1 \oplus x_2 \xrightarrow{x_2} x_1 \oplus x_2 \oplus y_1 \oplus y_2$ will be added to (V, E) by construction. Denote by V' the multiset of updated nodes after the query. Then, we require that $x_1 \oplus x_2 \oplus y_1 \oplus y_2$ does not occur twice in V'_{in} (in order to establish R1), and moreover that it does not occur in V'_{out} (in order to establish R2). If we define $V_{\mathrm{new}} = \{x_1 \oplus x'_2, x_1 \oplus x'_2 \oplus y_1 \oplus y'_2 \mid (x'_2, y'_2) \in Q\}$ to be the multiset of newly added nodes to V in the query to R_P, both requirements are satisfied if $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \notin V \cup (V_{\mathrm{new}} \backslash \{x_1 \oplus x_2 \oplus y_1 \oplus y_2\})$ holds for all $(x_2, y_2) \in Q$ such that $x_1 \oplus x_2 \in r(V)$. A similar condition can be derived for queries to S_Q. These requirements translate to lines 1k and 2e in the description of the simulator in Fig. 2.

Restricting the growth of $\bar{r}(V) \cap \mathrm{dom}(P)$

INVERSE QUERIES. As explained, S never increases $\bar{r}(V) \subseteq r(V)$ in inverse queries. Hence, requirement R3 is naturally satisfied. Furthermore, R4 is guaranteed if queries to $\mathsf{S}_{P^{-1}}$ are never answered with a node in $\bar{r}(V)$. This requirement translates to line 3c from Fig. 2.

FORWARD QUERIES. First consider requirement R3. Let the distinguisher make a query to S_P or S_Q, such that $\bar{r}(V)$ gets increased. By construction and the fact that requirement R2 is satisfied, this means that an edge $x_1 \oplus x_2 \xrightarrow{x_2} x_1 \oplus x_2 \oplus y_1 \oplus y_2$ is added to (V, E), such that $IV \xrightarrow{M} x_1 \oplus x_2$ for some $M \in \left(\mathbb{Z}_2^l\right)^*$, and $x_2 \in Z(M)$. The simulator needs to be designed such that the newly added value to $\bar{r}(V)$, $x_1 \oplus x_2 \oplus y_1 \oplus y_2$, is not a member of (the updated) $\mathrm{dom}(P)$. This requirement translates to lines 1l and 2f in Fig. 2. Requirement R4 is clearly not applicable to queries to S_Q. Consider a query x_1 to S_P, where $x_1 \in \bar{r}(V)$. Then, the simulator is forced to increase $\bar{r}(V) \cap \mathrm{dom}(P)$. As $x_1 \in \bar{r}(V)$, there exists an M such that $IV \xrightarrow{M} x_1$ and $\mathrm{depad}(M) \neq \perp$. The output of the simulator needs to be consistent with its random oracle, such that $RO(\mathrm{depad}(M)) = \mathrm{chop}_{l-n}(\mathsf{S}_P(x_1) \oplus x_1)$. This requirement translates to lines 1b-1e in the description of the simulator in Fig. 2.

4.3 Proof of Thm. 1

Thm. 1 will be proven via a game-playing argument, where the games are used to simulate one of the worlds (left or right). It is inspired by the proofs of [12], but differs in several aspects. Let S be the simulator of Fig. 2, and let $\mathcal{D}$ be any distinguisher that makes at most q_L left queries of maximal length $(K-1)l$ bits, where $K \geq 1$, q_P right queries to P and q_Q right queries to Q. Recall from Def. 1 that the goal is to bound:

$$\mathrm{Adv}^{\mathrm{pro}}_{\mathsf{Gr},\mathsf{S}}(\mathcal{D}) = \left| \Pr\left(\mathcal{D}^{\mathsf{Gr}^{P,Q},(P,Q)} = 1\right) - \Pr\left(\mathcal{D}^{RO,\mathsf{S}^{RO}} = 1\right) \right|. \tag{3}$$

Game 1 (Fig. 3). The left oracle L_1 of game 1 is a lazily-sampled random oracle, and the four interfaces of the right oracle are the simulator of Fig. 2, except for the inclusion of some failure conditions $\mathbf{bad}_i$ $(i = 0, \ldots, 4)$. In other words, we have $G_1 = (RO, \mathsf{S}^{RO})$, and in particular, $\Pr\left(\mathcal{D}^{RO,\mathsf{S}^{RO}} = 1\right) = \Pr\left(\mathcal{D}^{G_1} = 1\right)$.

Game 2 (Fig. 4). Game 2 only differs from game 1 in the left oracle: L_1 is replaced by a relay oracle L_2 that simply passes the queries made by the distinguisher to L_1. The right oracle remains unchanged, and still queries the subroutine L_1. The distinguisher has identical

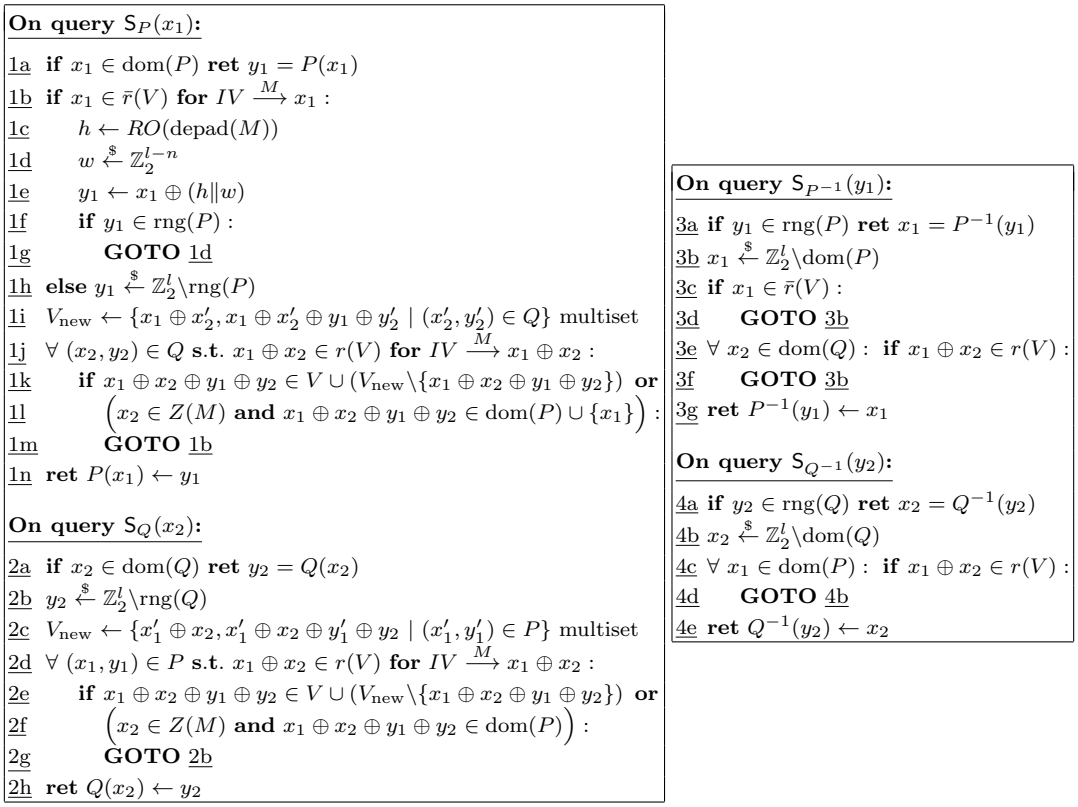

Fig. 2. The simulator S for P and Q used in the proof of Thm. 1.

views in G_1 and G_2. Formally, we obtain $\Pr\left(\mathcal{D}^{G_1} = 1\right) = \Pr\left(\mathcal{D}^{G_2} = 1\right)$.

Game 3 (Fig. 5). Game 3 differs from game 2 in the fact that the left oracle L_2 is replaced by the Grøstl hash function, which makes queries to the right oracle. The right oracle itself remains unchanged, and still queries subroutine L_1. It is proven in Prop. 2 that, until $\mathbf{bad} := \bigvee_{i=0}^{4} \mathbf{bad}_i$ occurs in any of the two games, both are identical. Formally, we obtain:

$$\left|\Pr\left(\mathcal{D}^{G_2} = 1\right) - \Pr\left(\mathcal{D}^{G_3} = 1\right)\right| \leq \Pr\left(\mathcal{D}^{G_2} \text{ sets } \mathbf{bad}\right) + \Pr\left(\mathcal{D}^{G_3} \text{ sets } \mathbf{bad}\right). \tag{4}$$

Game 4 (Fig. 6). Game 4 differs from game 3 in the fact that the right oracle does not query subroutine L_1 anymore, but rather, it generates the outcomes itself. Concretely, in line 1c, h is now randomly sampled from $\mathbb{Z}_2^n$. The distinguisher cannot notice the difference: as the padding rule is injective, in game 3 the right oracle R_P will never query its left oracle twice on the same value, and hence it will always receive $h \xleftarrow{\$} \mathbb{Z}_2^n$. Formally, we obtain $\Pr\left(\mathcal{D}^{G_3} = 1\right) = \Pr\left(\mathcal{D}^{G_4} = 1\right)$.

Game 5 (Fig. 6). Game 5 only differs from game 4 in the fact that the **GOTO**-statements are removed. In other words, game 5 and game 4 proceed identically until **bad** occurs. As a consequence:

$$\left|\Pr\left(\mathcal{D}^{G_4} = 1\right) - \Pr\left(\mathcal{D}^{G_5} = 1\right)\right| \leq \Pr\left(\mathcal{D}^{G_4} \text{ sets } \mathbf{bad}\right). \tag{5}$$

Game 6 (Fig. 7). The left oracle of game 6 is the Grøstl algorithm, and the four interfaces of the right oracle perfectly mimic two lazily-sampled random permutations P and Q. In

other words, we have $G_6 = (\mathsf{Gr}^{P,Q}, (P,Q))$, and thus $\Pr\left(\mathcal{D}^{G_6} = 1\right) = \Pr\left(\mathcal{D}^{\mathsf{Gr}^{P,Q},(P,Q)} = 1\right)$. The only difference between games 6 and 5 is in the forward queries to R_P: in game 5, some queries to R_P are answered with uniform random samples from $\mathbb{Z}_2^l$. Therefore, distinguishing game 6 from game 5 is at least as hard as distinguishing a random permutation from a random function. As R_P will be queried at most $q_P + (K+1)q_L =: r_P$ times, we obtain:

$$\left|\Pr\left(\mathcal{D}^{G_5} = 1\right) - \Pr\left(\mathcal{D}^{G_6} = 1\right)\right| \leq \frac{r_P^2}{2^l}. \tag{6}$$

Finally, observe that $\Pr\left(\mathcal{D}^{G_2} \text{ sets } \mathbf{bad}\right) \leq \Pr\left(\mathcal{D}^{G_3} \text{ sets } \mathbf{bad}\right) = \Pr\left(\mathcal{D}^{G_4} \text{ sets } \mathbf{bad}\right)$. Now, we conclude that (3) reduces to:

$$\mathrm{Adv}^{\mathrm{pro}}_{\mathsf{Gr},\mathsf{S}}(\mathcal{D}) \leq \frac{r_P^2}{2^l} + 3 \cdot \Pr\left(\mathcal{D}^{G_4} \text{ sets } \mathbf{bad}\right).$$

Game 7 (Fig. 8). Game 7 is used to simplify the computation of the probability that $\mathcal{D}^{G_4}$ sets **bad**. In game 7, the failure conditions for $\mathbf{bad}_0, \ldots, \mathbf{bad}_4$ of game 4 are rewritten into sets $A_0, \ldots, A_4$. By the straightforward definition of A_0, A_3 and A_4, it is clear that for $i = 0, 3, 4$, $\mathcal{D}^{G_4}$ sets $\mathbf{bad}_i$ if and only if $\mathcal{D}^{G_7}$ sets $\mathbf{bad}_i$. Now, suppose $\mathcal{D}^{G_4}$ sets $\mathbf{bad}_1$. This means that for some $(x_2, y_2) \in Q$ such that $x_1 \oplus x_2 \in r(V)$ either one of the following two cases occurred:

$$y_1 = \begin{cases} x_1 \oplus x_2 \oplus y_2 \oplus s, \text{ for some } s \in V \cup (V_{\mathrm{new}} \backslash \{x_1 \oplus x_2 \oplus y_1 \oplus y_2\}), \\ x_1 \oplus x_2 \oplus y_2 \oplus x_1', \text{ for some } x_1' \in \mathrm{dom}(P) \cup \{x_1\}. \end{cases}$$

By definition of A_1, this means that $y_1 \in A_1$. In other words, $\mathcal{D}^{G_7}$ sets $\mathbf{bad}_1$ if $\mathcal{D}^{G_4}$ sets $\mathbf{bad}_1$. A similar observation holds for $\mathbf{bad}_2$. As a consequence, $\Pr\left(\mathcal{D}^{G_4} \text{ sets } \mathbf{bad}\right) \leq \Pr\left(\mathcal{D}^{G_7} \text{ sets } \mathbf{bad}\right)$, and therefore we obtain:

$$\mathrm{Adv}^{\mathrm{pro}}_{\mathsf{Gr},\mathsf{S}}(\mathcal{D}) \leq \frac{r_P^2}{2^l} + 3 \cdot \Pr\left(\mathcal{D}^{G_7} \text{ sets } \mathbf{bad}_1 \mid \neg\mathbf{bad}_0\right) + 3 \sum_{\substack{i=0 \\ i \neq 1}}^{4} \Pr\left(\mathcal{D}^{G_7} \text{ sets } \mathbf{bad}_i\right). \tag{7}$$

In the remainder, we concentrate on the computation of these probabilities. Observe that the distinguisher makes at most $q_P + (K+1)q_L =: r_P$ queries to $R_P, R_{P^{-1}}$ and $q_Q + Kq_L =: r_Q$ queries to $R_Q, R_{Q^{-1}}$.

$\Pr\left(\mathcal{D}^{G_7} \textbf{ sets } \mathbf{bad}_0\right)$**.** Consider the j^{th} query to R_P, $1 \leq j \leq r_P$. The probability that $\mathbf{bad}_0$ is set in this query, $\mathbf{bad}_0^j$, equals the probability that y_1 hits A_0. But as y_1 is taken uniformly at random from a set of size 2^l, and A_0 is of size at most r_P, $\mathbf{bad}_0^j$ occurs with probability at most $\frac{r_P}{2^l}$. By the union bound, we obtain:

$$\Pr\left(\mathcal{D}^{G_7} \text{ sets } \mathbf{bad}_0\right) \leq \frac{r_P^2}{2^l}; \tag{8}$$

$\Pr\left(\mathcal{D}^{G_7} \textbf{ sets } \mathbf{bad}_1 \mid \mathcal{D}^{G_7} \textbf{ sets } \neg\mathbf{bad}_0\right)$**.** Consider the j^{th} query to R_P, $1 \leq j \leq r_P$. The probability that $\mathbf{bad}_1$ is set in this query, $\mathbf{bad}_1^j$, equals the probability that y_1 hits A_1. But as y_1 is taken uniformly at random from a set of size at least $2^l - r_P$ (because $\mathcal{D}^{G_7}$

sets $\neg\mathbf{bad}_0$), and A_1 is of size at most $r_Q(2r_Pr_Q + r_P)$, $\mathbf{bad}_1^j$ occurs with probability at most $\frac{r_Pr_Q(2r_Q+1)}{2^l-r_P}$. By the union bound, we obtain:

$$\Pr\left(\mathcal{D}^{G_7} \text{ sets } \mathbf{bad}_1 \mid \mathcal{D}^{G_7} \text{ sets } \neg\mathbf{bad}_0\right) \leq \frac{r_P^2 r_Q(2r_Q+1)}{2^l - r_P}; \tag{9}$$

$\Pr\left(\mathcal{D}^{G_7} \textbf{ sets } \mathbf{bad}_2\right)$**.** Consider the j^{th} query to R_Q, $1 \leq j \leq r_Q$. The probability that $\mathbf{bad}_2$ is set in this query, $\mathbf{bad}_2^j$, equals the probability that y_2 hits A_2. But as y_2 is taken uniformly at random from a set of size at least $2^l - r_Q$, and A_2 is of size at most $r_P(2r_Pr_Q + r_P)$, $\mathbf{bad}_2^j$ occurs with probability at most $\frac{r_P^2(2r_Q+1)}{2^l-r_Q}$. By the union bound, we obtain:

$$\Pr\left(\mathcal{D}^{G_7} \text{ sets } \mathbf{bad}_2\right) \leq \frac{r_P^2 r_Q(2r_Q+1)}{2^l - r_Q}; \tag{10}$$

$\Pr\left(\mathcal{D}^{G_7} \textbf{ sets } \mathbf{bad}_3\right)$**.** Consider the j^{th} query to $R_{P^{-1}}$, $1 \leq j \leq r_P$. The probability that $\mathbf{bad}_3$ is set in this query, $\mathbf{bad}_3^j$, equals the probability that x_1 hits A_3. But as x_1 is taken uniformly at random from a set of size at least $2^l - r_P$, and A_3 is of size at most $r_Pr_Q + r_Pr_Q^2$, $\mathbf{bad}_3^j$ occurs with probability at most $\frac{r_Pr_Q(r_Q+1)}{2^l-r_P}$. By the union bound, we obtain:

$$\Pr\left(\mathcal{D}^{G_7} \text{ sets } \mathbf{bad}_3\right) \leq \frac{r_P^2 r_Q(r_Q+1)}{2^l - r_P}; \tag{11}$$

$\Pr\left(\mathcal{D}^{G_7} \textbf{ sets } \mathbf{bad}_4\right)$**.** Consider the j^{th} query to $R_{Q^{-1}}$, $1 \leq j \leq r_Q$. The probability that $\mathbf{bad}_4$ is set in this query, $\mathbf{bad}_4^j$, equals the probability that x_2 hits A_4. But as x_2 is taken uniformly at random from a set of size at least $2^l - r_Q$, and A_4 is of size at most $r_P^2 r_Q$, $\mathbf{bad}_4^j$ occurs with probability at most $\frac{r_P^2 r_Q}{2^l-r_Q}$. By the union bound, we obtain:

$$\Pr\left(\mathcal{D}^{G_7} \text{ sets } \mathbf{bad}_4\right) \leq \frac{r_P^2 r_Q^2}{2^l - r_Q}. \tag{12}$$

Combining (7) and (8)-(12), under the assumption that $r_P, r_Q < 2^{l-1}$, we obtain:

$$\mathrm{Adv}^{\mathrm{pro}}_{\mathsf{Gr},\mathsf{S}}(\mathcal{D}) \leq \frac{58(q_P + (K+1)q_L)^2(q_Q + Kq_L)^2}{2^l}.$$

This completes the proof of Thm. 1.

Proposition 2. *Until* **bad** *occurs in either game 2 or game 3, both games are identical. Formally, we have* $\Pr\left(\mathcal{D}^{G_2} = 1 \;\middle|\; \mathcal{D}^{G_2} \textit{ sets } \neg\mathbf{bad}\right) = \Pr\left(\mathcal{D}^{G_3} = 1 \;\middle|\; \mathcal{D}^{G_3} \textit{ sets } \neg\mathbf{bad}\right)$.

Proof. We need to prove that the query outcomes in game 2 and 3 are identically distributed, until the distinguisher sets **bad** in either one of the games. As the right oracles of the games are the same, $\mathcal{D}$ can differentiate game 2 and 3 only if it discovers any inconsistencies in the answers by the left oracles (L_2 for game 2 and L_3 for game 3), given any list of queries made by $\mathcal{D}$ to the right oracle. Denote by P, Q the query history to the right oracles R_P, R_Q, and denote by (V, E) the graph defined by these queries (cf. Sect. 4.1). Denote any query history to L_i ($i = 2, 3$) by $\mathcal{L}$. Furthermore, denote by $\tilde{P}, \tilde{Q}$ the set of queries to the right oracles that are observed by the distinguisher[4], and denote by $(\tilde{V}, \tilde{E})$ the subgraph defined

[4] In game 3, the right oracles R_P, R_Q are also queried in each call to the left oracle, via lines 6d, 6e and 6g, but the distinguisher does not observe these queries.

by these. We focus on the outcomes of the left oracle: we need to prove that given the views $\tilde{P}, \tilde{Q}$, and given query history $\mathcal{L}$, the outcomes of new queries to the left oracle are identically distributed in game 2 and 3. Concretely, for $\alpha \in \mathbb{Z}_2^n$, we analyze the probability

$$\Pr\left(L_i(M) = \alpha \text{ in } G_i \mid \tilde{P}, \tilde{Q}, \mathcal{L};\ M \notin \mathrm{dom}(\mathcal{L});\ \mathcal{D}^{G_i} \text{ sets } \neg\mathbf{bad}\right). \tag{13}$$

Define $M' = (M'_1, \ldots, M'_k) = \mathrm{pad}(M)$ to be the padding of M. The query $L_i(M)$ is called 'evaluatable' by $\tilde{P}, \tilde{Q}$ if there exists an h_k in $\bar{r}(\tilde{V})$ such that $IV \xrightarrow{M'} h_k$, and $h_k \in \mathrm{dom}(\tilde{P})$. We will show that for both games the following holds: if $L_i(M)$ is evaluatable by $\tilde{P}, \tilde{Q}$, the query answer can be obtained deterministically from this history. On the other hand, if it is *not* evaluatable by $\tilde{P}, \tilde{Q}$, (13) holds with probability $1/2^n$ only. In other words, this probability is the same in both games $i = 2, 3$, which proves the claim that the answers by L_2, L_3 are identically distributed.
For the purpose of the proof, we also consider evaluatability by P, Q, which is defined similarly as before. Observe that $H_i(M)$ is evaluatable by P, Q if it is evaluatable by $\tilde{P}, \tilde{Q}$. We now analyze (13). First we consider the case $L_i(M)$ is evaluatable by $\tilde{P}, \tilde{Q}$. Then we consider the case it is not evaluatable by these views (but it may be evaluatable by P, Q).

(1) $L_i(M)$ **$(i = 2, 3)$ is evaluatable by $\tilde{P}, \tilde{Q}$.** In both games, this means that there exists an h_k in $\bar{r}(\tilde{V})$ such that $IV \xrightarrow{M'} h_k$, and $h_k \in \mathrm{dom}(\tilde{P})$. By Claim 2 below, there are no colliding paths and in particular the described path M' is unique. Furthermore, due to Claim 3 below, h_k had been added to $\mathrm{dom}(\tilde{P})$ in a forward query, *after* it was added to $\bar{r}(\tilde{V})$. Therefore, by line 1c, we have $R_P(h_k) = h_k \oplus (h \| w)$, where $h = L_1(M)$. As a consequence, $L_1(M)$, and thus $L_2(M)$ and $L_3(M)$, is fully determined by $\tilde{P}, \tilde{Q}$, which means that the outcomes in game 2 and 3 are identically distributed;
(2) $L_i(M)$ **$(i = 2, 3)$ is not evaluatable by $\tilde{P}, \tilde{Q}$, but it is evaluatable by P, Q.** This event is excluded for game 2 as $(\tilde{P}, \tilde{Q}) = (P, Q)$ in this game. In game 3, P, Q also includes queries made to the right oracle via the left oracle L_3. We will show, however, that (13) holds with probability $1/2^n$ then. Similarly to case (1), there exists an h_k in $\bar{r}(V) \cap \mathrm{dom}(P)$ such that $IV \xrightarrow{M'} h_k$ and $R_P(h_k) = h_k \oplus (h \| w)$, where $h = L_1(M)$. But $L_3(M)$ is *not* evaluatable by $\tilde{P}, \tilde{Q}$, which means that h_k had been queried to R_P independently of $\tilde{P}, \tilde{Q}$. Furthermore, $L_3(M)$ is also independent of $\mathcal{L}$.[5] Concluding, (13) holds with probability $1/2^n$ in this case;
(3) $L_i(M)$ **$(i = 2, 3)$ is not evaluatable by P, Q.** As a consequence, there either exists *no* $h_k \in \bar{r}(V)$ such that $IV \xrightarrow{M'} h_k$, or there exists such h_k, but it is no element of $\mathrm{dom}(P)$. For game 2, as $M \notin \mathrm{dom}(\mathcal{L})$ this implies that M had not been queried to L_1 before (L_1 is queried in lines 6a and 1c only). Therefore, in this case $L_2(M)$ outputs a value h randomly sampled from $\mathbb{Z}_2^n$. For game 3, let $j \leq k$ be the maximal index such that $IV = h_0 \xrightarrow{M'_1} \cdots \xrightarrow{M'_j} h_j$ is a path in (V, E). We consider the following cases:
   (i) $j = k$. Then, there exists an $h_k \in \bar{r}(V)$ such that $IV \xrightarrow{M'} h_k$, but as $L_3(M)$ is not evaluatable, we have $h_k \notin \mathrm{dom}(P)$. In line 6h of the oracle query of $L_3(M)$,

[5] Observe that in game 3, $\mathcal{L}$ consists of pairs $(\bar{M}, \bar{h})$ such that $\bar{h} = \mathrm{chop}_{l-n}(R_P(\bar{h}_k) \oplus \bar{h}_k)$ for some $\bar{h}_k \in \bar{r}(V) \cap \mathrm{dom}(P)$, where, by Claim 3, $R_P(\bar{h}_k)$ had been generated via lines 1b-1e. As there are no colliding paths in (V, E) by Claim 2, h_k differs from all such $\bar{h}_k$'s, and in particular $\mathcal{L}$ reveals nothing about $L_3(M)$.

$R_P(h_k)$ will then be computed via lines 1b-1e: $R_P(h_k) = h_k \oplus (h \| w)$ for $h \xleftarrow{\$} \mathbb{Z}_2^n$. The outcome $L_3(M)$ thus equals $L_3(M) = \mathrm{chop}_{l-n}(R_P(h_k) \oplus h_k) = h$. As a consequence, the outcomes of L_2 and L_3 are identically distributed in this case;

(ii) $j < k$. Then, there exists a path $IV \to h_j$ labeled by $(M'_1, \ldots, M'_j)$, but (V, E) contains no edge $h_j \to h_{j+1}$ labeled by M'_{j+1}. By virtue of Claim 2, in the $(j+1)^{\text{th}}$ iteration of lines 6c-6f, a new node h_{j+1} will be added to $r(V)$ such that h_{j+1} was not rooted yet and there is no outgoing edge from h_{j+1} in the updated graph. The same holds for all subsequent iterations, and in particular h_k will be newly added to $\bar{r}(V)$ in the k^{th} iteration. Due to Claim 3, this newly added note is not an element of $\mathrm{dom}(P)$ after this last round. Now, the same analysis as in (3i) applies. □

Claim 2. Suppose $\mathcal{D}^{G_i}$ sets $\neg\mathbf{bad}$ (for $i = 2, 3$). Consider a node $s \in r(V)$, and a right oracle query in which an edge (s, t) will be added to (V, E). Denote by (V', E') the updated graph (after the query). Then, t has no incoming or outgoing edge in $(V', E' \backslash \{(s, t)\})$.
As a consequence, after the execution of G_i, the final graph contains no colliding paths.

Proof. In a right query to $R_{P^{-1}}$ or $R_{Q^{-1}}$, none of the newly added edges have a rooted node as starting point, by $\neg(\mathbf{bad}_3 \vee \mathbf{bad}_4)$ (lines 3f and 4c). Consider a query x_1 to R_P, and let (V, E) be the graph before the query. An outgoing edge from $s \in r(V)$ will only be added if $s = x_1 \oplus x_2$ for some $x_2 \in \mathrm{dom}(Q)$. By construction, the end node of the edge is $x_1 \oplus x_2 \oplus y_1 \oplus y_2 =: t$. By line 1l and $\neg\mathbf{bad}_1$, we have (a) $t \notin V$, (b) none of the newly added edges will leave from t and (c) apart from (s, t), none of the newly added edges will arrive at t. As a consequence, t is an isolated node in $(V', E' \backslash \{(s, t)\})$. A similar argument holds for queries to R_Q, by line 2e and $\neg\mathbf{bad}_2$.
We prove that the final graph contains no colliding paths by mathematical induction. Before the first query is made, $E = \emptyset$ and hence no colliding paths occur. Assume (V, E) contains no colliding paths and consider a right oracle query. We can sequentially apply the above reasoning and discard all newly added edges (s, t) for $s \in r(V)$, in order to observe that colliding paths in (V', E') imply colliding paths in (V, E). By the induction hypothesis, these do not occur. □

Claim 3. Suppose $\mathcal{D}^{G_i}$ sets $\neg\mathbf{bad}$ (for $i = 2, 3$). Consider a right oracle query in which a node t will be added to $\bar{r}(V)$. Then, t is no element of (the updated) $\mathrm{dom}(P)$. Furthermore, $\bar{r}(V) \cap \mathrm{dom}(P)$ will only be increased in forward queries to R_P.

Proof. As a direct consequence of Claim 2, $\bar{r}(V)$ will be increased only if an edge $x_1 \oplus x_2 \xrightarrow{x_2} x_1 \oplus x_2 \oplus y_1 \oplus y_2$ is added such that $IV \xrightarrow{M} x_1 \oplus x_2$ is a path in (V, E), and $x_2 \in Z(M)$. Due to lines 1m and 2f, and by $\neg(\mathbf{bad}_1 \vee \mathbf{bad}_2)$, this newly added node is not an element of (the updated) $\mathrm{dom}(P)$. Furthermore, an inverse query to R_P will never be answered with a node already in $\bar{r}(V)$, by line 3c and $\neg\mathbf{bad}_3$, and therefore $\bar{r}(V) \cap \mathrm{dom}(P)$ will only be increased in forward queries to P. □

References

[1] Elena Andreeva, Gregory Neven, Bart Preneel, and Thomas Shrimpton. Seven-property-preserving iterated hashing: ROX. In *Advances in Cryptology - ASIACRYPT '07*, volume 4833 of *Lecture Notes in Computer Science*, pages 130–146, Berlin, 2007. Springer-Verlag.

[2] Mihir Bellare and Thomas Ristenpart. Multi-property-preserving hash domain extension and the EMD Transform. In *Advances in Cryptology - ASIACRYPT '06*, volume 4284 of *Lecture Notes in Computer Science*, pages 299–314, Berlin, 2006. Springer-Verlag.

[3] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, New York, 1993. ACM.

[4] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles van Assche. On the indifferentiability of the sponge construction. In *Advances in Cryptology - EUROCRYPT '08*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197, Berlin, 2008. Springer-Verlag.

[5] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions, ECRYPT Hash Workshop 2007. `http://sponge.noekeon.org/SpongeFunctions.pdf`.

[6] Rishiraj Bhattacharyya, Avradip Mandal, and Mridul Nandi. Security analysis of the mode of JH hash function. In *Fast Software Encryption '10*, Lecture Notes in Computer Science, Berlin, 2010. Springer-Verlag.

[7] Eli Biham and Orr Dunkelman. A framework for iterative hash functions – HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007. `http://eprint.iacr.org/2007/278`.

[8] Emmanuel Bresson, Anne Canteaut, Benoît Chevallier-Mames, Christophe Clavier, Thomas Fuhr, Aline Gouget, Thomas Icart, Jean-François Misarsky, Mara Naya-Plasencia, Pascal Paillier, Thomas Pornin, Jean-René Reinhard, Céline Thuillet, and Marion Videau. Shabal, a Submission to NIST's Cryptographic Hash Algorithm Competition, 2009.

[9] Emmanuel Bresson, Anne Canteaut, Benoît Chevallier-Mames, Christophe Clavier, Thomas Fuhr, Aline Gouget, Thomas Icart, Jean-François Misarsky, Mara Naya-Plasencia, Pascal Paillier, Thomas Pornin, Jean-René Reinhard, Céline Thuillet, and Marion Videau. Indifferentiability with distinguishers: Why shabal does not require ideal ciphers. Cryptology ePrint Archive, Report 2009/199, 2009. `http://eprint.iacr.org/2009/199`.

[10] Donghoon Chang, Sangjin Lee, Mridul Nandi, and Moti Yung. Indifferentiable security analysis of popular hash functions with prefix-free padding. In *Advances in Cryptology - ASIACRYPT '06*, volume 4284 of *Lecture Notes in Computer Science*, pages 283–298, Berlin, 2006. Springer-Verlag.

[11] Donghoon Chang and Mridul Nandi. Improved indifferentiability security analysis of chopMD hash function. In *Fast Software Encryption '08*, volume 5086 of *Lecture Notes in Computer Science*, pages 429–443, Berlin, 2008. Springer-Verlag.

[12] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In *Advances in Cryptology - CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448, Berlin, 2005. Springer-Verlag.

[13] Ivan Damgård. A design principle for hash functions. In *Advances in Cryptology - CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427, Berlin, 1990. Springer-Verlag.

[14] Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the CBC, cascade and HMAC modes. In *Advances in Cryptology - CRYPTO '04*, volume 3152 of *Lecture Notes in Computer Science*, pages 494–510, Berlin, 2004. Springer-Verlag.

[15] Yevgeniy Dodis, Krzysztof Pietrzak, and Prashant Puniya. A new mode of operation for block ciphers and length-preserving MACs. In *Advances in Cryptology - EUROCRYPT '08*, volume 4965 of *Lecture Notes in Computer Science*, pages 198–219, Berlin, 2008. Springer-Verlag.

[16] Yevgeniy Dodis, Thomas Ristenpart, and Thomas Shrimpton. Salvaging Merkle-Damgård for practical applications. In *Advances in Cryptology - EUROCRYPT '09*, volume 5479 of *Lecture Notes in Computer Science*, pages 371–388, Berlin, 2009. Springer-Verlag.

[17] Praveen Gauravaram, Lars Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren Thomsen. Grøstl – a SHA-3 candidate, 2009. `http://www.groestl.info`.

[18] Shoichi Hirose, Je Hong Park, and Aaram Yun. A simple variant of the Merkle-Damgård scheme with a permutation. In *Advances in Cryptology - ASIACRYPT '07*, volume 4833 of *Lecture Notes in Computer Science*, pages 113–129, Berlin, 2007. Springer-Verlag.

[19] Stefan Lucks. A failure-friendly design principle for hash functions. In *Advances in Cryptology - ASIACRYPT '05*, volume 3788 of *Lecture Notes in Computer Science*, pages 474–494, Berlin, 2005. Springer-Verlag.

[20] Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *Theory of Cryptography Conference '04*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39, Berlin, 2004. Springer-Verlag.

[21] Ralph Merkle. One way hash functions and DES. In *Advances in Cryptology - CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446, Berlin, 1990. Springer-Verlag.

[22] Shoji Miyaguchi, Kazuo Ohta, and Masahiko Iwata. Confirmation that some hash functions are not collision free. In *Advances in Cryptology - EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 326–343, Berlin, 1990. Springer-Verlag.

[23] National Institute for Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA3) Family, November 2007. `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`.

[24] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *Advances in Cryptology - CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378, Berlin, 1993. Springer-Verlag.

[25] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology - CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36, Berlin, 2005. Springer-Verlag.

[26] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *Advances in Cryptology - EUROCRYPT '05*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35, Berlin, 2005. Springer-Verlag.

A Games Used in the Proof of Thm. 1

On right query $R_P(x_1)$:

1a **if** $x_1 \in \mathrm{dom}(P)$ **ret** $y_1 = P(x_1)$
1b **if** $x_1 \in \bar{r}(V)$ **for** $IV \xrightarrow{M} x_1$:
1c $h \leftarrow L_1(\mathrm{depad}(M))$
1d $w \xleftarrow{\$} \mathbb{Z}_2^{l-n}$
1e $y_1 \leftarrow x_1 \oplus (h \| w)$
1f **if** $y_1 \in \mathrm{rng}(P)$:
1g $\mathbf{bad}_0 \leftarrow \mathbf{true}$
1h **GOTO** 1d
1i **else** $y_1 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{rng}(P)$
1j $V_{\mathrm{new}} \leftarrow \{x_1 \oplus x_2', x_1 \oplus x_2' \oplus y_1 \oplus y_2' \mid (x_2', y_2') \in Q\}$ multiset
1k $\forall\, (x_2, y_2) \in Q$ **s.t.** $x_1 \oplus x_2 \in r(V)$ **for** $IV \xrightarrow{M} x_1 \oplus x_2$:
1l **if** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in V \cup (V_{\mathrm{new}} \backslash \{x_1 \oplus x_2 \oplus y_1 \oplus y_2\})$ **or**
1m $\Big(x_2 \in Z(M)$ **and** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in \mathrm{dom}(P) \cup \{x_1\}\Big)$:
1n $\mathbf{bad}_1 \leftarrow \mathbf{true}$
1o **GOTO** 1b
1p **ret** $P(x_1) \leftarrow y_1$

On right query $R_Q(x_2)$:

2a **if** $x_2 \in \mathrm{dom}(Q)$ **ret** $y_2 = Q(x_2)$
2b $y_2 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{rng}(Q)$
2c $V_{\mathrm{new}} \leftarrow \{x_1' \oplus x_2, x_1' \oplus x_2 \oplus y_1' \oplus y_2 \mid (x_1', y_1') \in P\}$ multiset
2d $\forall\, (x_1, y_1) \in P$ **s.t.** $x_1 \oplus x_2 \in r(V)$ **for** $IV \xrightarrow{M} x_1 \oplus x_2$:
2e **if** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in V \cup (V_{\mathrm{new}} \backslash \{x_1 \oplus x_2 \oplus y_1 \oplus y_2\})$ **or**
2f $\Big(x_2 \in Z(M)$ **and** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in \mathrm{dom}(P)\Big)$:
2g $\mathbf{bad}_2 \leftarrow \mathbf{true}$
2h **GOTO** 2b
2i **ret** $Q(x_2) \leftarrow y_2$

On right query $R_{P^{-1}}(y_1)$:

3a **if** $y_1 \in \mathrm{rng}(P)$ **ret** $x_1 = P^{-1}(y_1)$
3b $x_1 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{dom}(P)$
3c **if** $x_1 \in \bar{r}(V)$:
3d $\mathbf{bad}_3 \leftarrow \mathbf{true}$
3e **GOTO** 3b
3f $\forall\, x_2 \in \mathrm{dom}(Q)$: **if** $x_1 \oplus x_2 \in r(V)$:
3g $\mathbf{bad}_3 \leftarrow \mathbf{true}$
3h **GOTO** 3b
3i **ret** $P^{-1}(y_1) \leftarrow x_1$

On right query $R_{Q^{-1}}(y_2)$:

4a **if** $y_2 \in \mathrm{rng}(Q)$ **ret** $x_2 = Q^{-1}(y_2)$
4b $x_2 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{dom}(Q)$
4c $\forall\, x_1 \in \mathrm{dom}(P)$: **if** $x_1 \oplus x_2 \in r(V)$:
4d $\mathbf{bad}_4 \leftarrow \mathbf{true}$
4e **GOTO** 4b
4f **ret** $Q^{-1}(y_2) \leftarrow x_2$

On left query $L_1(M)$:

5a **if** $M \in \mathrm{dom}(\mathcal{H})$ **ret** $h = \mathcal{H}(M)$
5b $h \xleftarrow{\$} \mathbb{Z}_2^n$
5c **ret** $\mathcal{H}(M) \leftarrow h$

Fig. 3. Game 1. The distinguisher has access to L_1, R^{L_1}.

On right query $R_P(x_1)$:

1a **if** $x_1 \in \mathrm{dom}(P)$ **ret** $y_1 = P(x_1)$
1b **if** $x_1 \in \bar{r}(V)$ **for** $IV \xrightarrow{M} x_1$:
1c $h \leftarrow L_1(\mathrm{depad}(M))$
1d $w \xleftarrow{\$} \mathbb{Z}_2^{l-n}$
1e $y_1 \leftarrow x_1 \oplus (h \| w)$
1f **if** $y_1 \in \mathrm{rng}(P)$:
1g **bad**$_0 \leftarrow$ **true**
1h **GOTO** 1d
1i **else** $y_1 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{rng}(P)$
1j $V_{\mathrm{new}} \leftarrow \{x_1 \oplus x_2', x_1 \oplus x_2' \oplus y_1 \oplus y_2' \mid (x_2', y_2') \in Q\}$ multiset
1k $\forall\ (x_2, y_2) \in Q$ **s.t.** $x_1 \oplus x_2 \in r(V)$ **for** $IV \xrightarrow{M} x_1 \oplus x_2$:
1l **if** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in V \cup (V_{\mathrm{new}} \backslash \{x_1 \oplus x_2 \oplus y_1 \oplus y_2\})$ **or**
1m $\Big(x_2 \in Z(M)$ **and** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in \mathrm{dom}(P) \cup \{x_1\}\Big)$:
1n **bad**$_1 \leftarrow$ **true**
1o **GOTO** 1b
1p **ret** $P(x_1) \leftarrow y_1$

On right query $R_Q(x_2)$:

2a **if** $x_2 \in \mathrm{dom}(Q)$ **ret** $y_2 = Q(x_2)$
2b $y_2 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{rng}(Q)$
2c $V_{\mathrm{new}} \leftarrow \{x_1' \oplus x_2, x_1' \oplus x_2 \oplus y_1' \oplus y_2 \mid (x_1', y_1') \in P\}$ multiset
2d $\forall\ (x_1, y_1) \in P$ **s.t.** $x_1 \oplus x_2 \in r(V)$ **for** $IV \xrightarrow{M} x_1 \oplus x_2$:
2e **if** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in V \cup (V_{\mathrm{new}} \backslash \{x_1 \oplus x_2 \oplus y_1 \oplus y_2\})$ **or**
2f $\Big(x_2 \in Z(M)$ **and** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in \mathrm{dom}(P)\Big)$:
2g **bad**$_2 \leftarrow$ **true**
2h **GOTO** 2b
2i **ret** $Q(x_2) \leftarrow y_2$

On right query $R_{P^{-1}}(y_1)$:

3a **if** $y_1 \in \mathrm{rng}(P)$ **ret** $x_1 = P^{-1}(y_1)$
3b $x_1 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{dom}(P)$
3c **if** $x_1 \in \bar{r}(V)$:
3d **bad**$_3 \leftarrow$ **true**
3e **GOTO** 3b
3f $\forall\ x_2 \in \mathrm{dom}(Q)$: **if** $x_1 \oplus x_2 \in r(V)$:
3g **bad**$_3 \leftarrow$ **true**
3h **GOTO** 3b
3i **ret** $P^{-1}(y_1) \leftarrow x_1$

On right query $R_{Q^{-1}}(y_2)$:

4a **if** $y_2 \in \mathrm{rng}(Q)$ **ret** $x_2 = Q^{-1}(y_2)$
4b $x_2 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{dom}(Q)$
4c $\forall\ x_1 \in \mathrm{dom}(P)$: **if** $x_1 \oplus x_2 \in r(V)$:
4d **bad**$_4 \leftarrow$ **true**
4e **GOTO** 4b
4f **ret** $Q^{-1}(y_2) \leftarrow x_2$

Subroutine $L_1(M)$:

5a **if** $M \in \mathrm{dom}(\mathcal{H})$ **ret** $h = \mathcal{H}(M)$
5b $h \xleftarrow{\$} \mathbb{Z}_2^n$
5c **ret** $\mathcal{H}(M) \leftarrow h$

On left query $L_2(M)$:

6a **ret** $h \leftarrow L_1(M)$

Fig. 4. Game 2. The distinguisher has access to $L_2^{L_1}, R^{L_1}$.

On right query $R_P(x_1)$:

1a **if** $x_1 \in \mathrm{dom}(P)$ **ret** $y_1 = P(x_1)$
1b **if** $x_1 \in \bar{r}(V)$ **for** $IV \xrightarrow{M} x_1$:
1c $h \leftarrow L_1(\mathrm{depad}(M))$
1d $w \xleftarrow{\$} \mathbb{Z}_2^{l-n}$
1e $y_1 \leftarrow x_1 \oplus (h \| w)$
1f **if** $y_1 \in \mathrm{rng}(P)$:
1g **bad**$_0 \leftarrow$ **true**
1h **GOTO** 1d
1i **else** $y_1 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{rng}(P)$
1j $V_{\mathrm{new}} \leftarrow \{x_1 \oplus x_2', x_1 \oplus x_2' \oplus y_1 \oplus y_2' \mid (x_2', y_2') \in Q\}$ multiset
1k $\forall\ (x_2, y_2) \in Q$ **s.t.** $x_1 \oplus x_2 \in r(V)$ **for** $IV \xrightarrow{M} x_1 \oplus x_2$:
1l **if** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in V \cup (V_{\mathrm{new}} \backslash \{x_1 \oplus x_2 \oplus y_1 \oplus y_2\})$ **or**
1m $\Big(x_2 \in Z(M)$ **and** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in \mathrm{dom}(P) \cup \{x_1\}\Big)$:
1n **bad**$_1 \leftarrow$ **true**
1o **GOTO** 1b
1p **ret** $P(x_1) \leftarrow y_1$

On right query $R_Q(x_2)$:

2a **if** $x_2 \in \mathrm{dom}(Q)$ **ret** $y_2 = Q(x_2)$
2b $y_2 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{rng}(Q)$
2c $V_{\mathrm{new}} \leftarrow \{x_1' \oplus x_2, x_1' \oplus x_2 \oplus y_1' \oplus y_2 \mid (x_1', y_1') \in P\}$ multiset
2d $\forall\ (x_1, y_1) \in P$ **s.t.** $x_1 \oplus x_2 \in r(V)$ **for** $IV \xrightarrow{M} x_1 \oplus x_2$:
2e **if** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in V \cup (V_{\mathrm{new}} \backslash \{x_1 \oplus x_2 \oplus y_1 \oplus y_2\})$ **or**
2f $\Big(x_2 \in Z(M)$ **and** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in \mathrm{dom}(P)\Big)$:
2g **bad**$_2 \leftarrow$ **true**
2h **GOTO** 2b
2i **ret** $Q(x_2) \leftarrow y_2$

On right query $R_{P^{-1}}(y_1)$:

3a **if** $y_1 \in \mathrm{rng}(P)$ **ret** $x_1 = P^{-1}(y_1)$
3b $x_1 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{dom}(P)$
3c **if** $x_1 \in \bar{r}(V)$:
3d **bad**$_3 \leftarrow$ **true**
3e **GOTO** 3b
3f $\forall\ x_2 \in \mathrm{dom}(Q)$: **if** $x_1 \oplus x_2 \in r(V)$:
3g **bad**$_3 \leftarrow$ **true**
3h **GOTO** 3b
3i **ret** $P^{-1}(y_1) \leftarrow x_1$

On right query $R_{Q^{-1}}(y_2)$:

4a **if** $y_2 \in \mathrm{rng}(Q)$ **ret** $x_2 = Q^{-1}(y_2)$
4b $x_2 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{dom}(Q)$
4c $\forall\ x_1 \in \mathrm{dom}(P)$: **if** $x_1 \oplus x_2 \in r(V)$:
4d **bad**$_4 \leftarrow$ **true**
4e **GOTO** 4b
4f **ret** $Q^{-1}(y_2) \leftarrow x_2$

Subroutine $L_1(M)$:

5a **if** $M \in \mathrm{dom}(\mathcal{H})$ **ret** $h = \mathcal{H}(M)$
5b $h \xleftarrow{\$} \mathbb{Z}_2^n$
5c **ret** $\mathcal{H}(M) \leftarrow h$

On left query $L_3(M)$:

6a $(M_1', \ldots, M_k') \leftarrow \mathrm{pad}(M)$
6b $h_0 \leftarrow IV_n$
6c **for** $i = 1, \ldots, k$:
6d $a \leftarrow R_Q(M_i')$
6e $b \leftarrow R_P(h_{i-1} \oplus M_i')$
6f $h_i \leftarrow a \oplus b \oplus h_{i-1}$
6g $d \leftarrow R_P(h_k)$
6h $h \leftarrow \mathrm{chop}_{l-n}(d \oplus h_k)$
6i **ret** h

Fig. 5. Game 3. The distinguisher has access to $L_3^{R^{L_1}}, R^{L_1}$.

On right query $R_P(x_1)$:

1a **if** $x_1 \in \mathrm{dom}(P)$ **ret** $y_1 = P(x_1)$
1b **if** $x_1 \in \bar{r}(V)$:
1c $\quad h \xleftarrow{\$} \mathbb{Z}_2^n$
1d $\quad w \xleftarrow{\$} \mathbb{Z}_2^{l-n}$
1e $\quad y_1 \leftarrow x_1 \oplus (h \| w)$
1f $\quad$ **if** $y_1 \in \mathrm{rng}(P)$:
1g $\quad\quad$ **bad**$_0 \leftarrow$ **true**
1h $\quad\quad$ **GOTO** 1d
1i **else** $y_1 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{rng}(P)$
1j $V_{\mathrm{new}} \leftarrow \{x_1 \oplus x_2', x_1 \oplus x_2' \oplus y_1 \oplus y_2' \mid (x_2', y_2') \in Q\}$ multiset
1k $\forall\, (x_2, y_2) \in Q$ **s.t.** $x_1 \oplus x_2 \in r(V)$ **for** $IV \xrightarrow{M} x_1 \oplus x_2$:
1l $\quad$ **if** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in V \cup (V_{\mathrm{new}} \backslash \{x_1 \oplus x_2 \oplus y_1 \oplus y_2\})$ **or**
1m $\quad\quad \Big(x_2 \in Z(M)$ **and** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in \mathrm{dom}(P) \cup \{x_1\}\Big)$:
1n $\quad\quad$ **bad**$_1 \leftarrow$ **true**
1o $\quad\quad$ **GOTO** 1b
1p **ret** $P(x_1) \leftarrow y_1$

On right query $R_Q(x_2)$:

2a **if** $x_2 \in \mathrm{dom}(Q)$ **ret** $y_2 = Q(x_2)$
2b $y_2 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{rng}(Q)$
2c $V_{\mathrm{new}} \leftarrow \{x_1' \oplus x_2, x_1' \oplus x_2 \oplus y_1' \oplus y_2 \mid (x_1', y_1') \in P\}$ multiset
2d $\forall\, (x_1, y_1) \in P$ **s.t.** $x_1 \oplus x_2 \in r(V)$ **for** $IV \xrightarrow{M} x_1 \oplus x_2$:
2e $\quad$ **if** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in V \cup (V_{\mathrm{new}} \backslash \{x_1 \oplus x_2 \oplus y_1 \oplus y_2\})$ **or**
2f $\quad\quad \Big(x_2 \in Z(M)$ **and** $x_1 \oplus x_2 \oplus y_1 \oplus y_2 \in \mathrm{dom}(P)\Big)$:
2g $\quad\quad$ **bad**$_2 \leftarrow$ **true**
2h $\quad\quad$ **GOTO** 2b
2i **ret** $Q(x_2) \leftarrow y_2$

On right query $R_{P^{-1}}(y_1)$:

3a **if** $y_1 \in \mathrm{rng}(P)$ **ret** $x_1 = P^{-1}(y_1)$
3b $x_1 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{dom}(P)$
3c **if** $x_1 \in \bar{r}(V)$:
3d $\quad$ **bad**$_3 \leftarrow$ **true**
3e $\quad$ **GOTO** 3b
3f $\forall\, x_2 \in \mathrm{dom}(Q)$: **if** $x_1 \oplus x_2 \in r(V)$:
3g $\quad$ **bad**$_3 \leftarrow$ **true**
3h $\quad$ **GOTO** 3b
3i **ret** $P^{-1}(y_1) \leftarrow x_1$

On right query $R_{Q^{-1}}(y_2)$:

4a **if** $y_2 \in \mathrm{rng}(Q)$ **ret** $x_2 = Q^{-1}(y_2)$
4b $x_2 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{dom}(Q)$
4c $\forall\, x_1 \in \mathrm{dom}(P)$: **if** $x_1 \oplus x_2 \in r(V)$:
4d $\quad$ **bad**$_4 \leftarrow$ **true**
4e $\quad$ **GOTO** 4b
4f **ret** $Q^{-1}(y_2) \leftarrow x_2$

On left query $L_3(M)$:

5a $(M_1', \ldots, M_k') \leftarrow \mathrm{pad}(M)$
5b $h_0 \leftarrow IV_n$
5c **for** $i = 1, \ldots, k$:
5d $\quad a \leftarrow R_Q(M_i')$
5e $\quad b \leftarrow R_P(h_{i-1} \oplus M_i')$
5f $\quad h_i \leftarrow a \oplus b \oplus h_{i-1}$
5g $d \leftarrow R_P(h_k)$
5h $h \leftarrow \mathrm{chop}_{l-n}(d \oplus h_k)$
5i **ret** h

Fig. 6. Game 4 (including the boxed statements) and game 5 (with the boxed statements removed). In both games, the distinguisher has access to L_3^R, R.

On right query $R_P(x_1)$:

1a **if** $x_1 \in \mathrm{dom}(P)$ **ret** $y_1 = P(x_1)$
1b $y_1 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{rng}(P)$
1c **ret** $P(x_1) \leftarrow y_1$

On right query $R_Q(x_2)$:

2a **if** $x_2 \in \mathrm{dom}(Q)$ **ret** $y_2 = Q(x_2)$
2b $y_2 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{rng}(Q)$
2c **ret** $Q(x_2) \leftarrow y_2$

On right query $R_{P^{-1}}(y_1)$:

3a **if** $y_1 \in \mathrm{rng}(P)$ **ret** $x_1 = P^{-1}(y_1)$
3b $x_1 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{dom}(P)$
3c **ret** $P^{-1}(y_1) \leftarrow x_1$

On right query $R_{Q^{-1}}(y_2)$:

4a **if** $y_2 \in \mathrm{rng}(Q)$ **ret** $x_2 = Q^{-1}(y_2)$
4b $x_2 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{dom}(Q)$
4c **ret** $Q^{-1}(y_2) \leftarrow x_2$

On left query $L_3(M)$:

5a $(M_1', \ldots, M_k') \leftarrow \mathrm{pad}(M)$
5b $h_0 \leftarrow IV_n$
5c **for** $i = 1, \ldots, k$:
5d $\quad a \leftarrow R_Q(M_i')$
5e $\quad b \leftarrow R_P(h_{i-1} \oplus M_i')$
5f $\quad h_i \leftarrow a \oplus b \oplus h_{i-1}$
5g $d \leftarrow R_P(h_k)$
5h $h \leftarrow \mathrm{chop}_{l-n}(d \oplus h_k)$
5i **ret** h

Fig. 7. Game 6. The distinguisher has access to L_3^R, R.

On right query $R_P(x_1)$:

1a **if** $x_1 \in \mathrm{dom}(P)$ **ret** $y_1 = P(x_1)$
1b **if** $x_1 \in \bar{r}(V)$:
1c $\quad h \xleftarrow{\$} \mathbb{Z}_2^n$
1d $\quad w \xleftarrow{\$} \mathbb{Z}_2^{l-n}$
1e $\quad y_1 \leftarrow x_1 \oplus (h \| w)$
1f $\quad$ **if** $y_1 \in A_0$:
1g $\quad\quad$ **bad**$_0 \leftarrow$ **true**
1h $\quad\quad$ **GOTO** 1d
1i **else** $y_1 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{rng}(P)$
1j **if** $y_1 \in A_1$:
1k $\quad$ **bad**$_1 \leftarrow$ **true**
1l $\quad$ **GOTO** 1b
1m **ret** $P(x_1) \leftarrow y_1$

On right query $R_Q(x_2)$:

2a **if** $x_2 \in \mathrm{dom}(Q)$ **ret** $y_2 = Q(x_2)$
2b $y_2 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{rng}(Q)$
2c **if** $y_2 \in A_2$:
2d $\quad$ **bad**$_2 \leftarrow$ **true**
2e $\quad$ **GOTO** 2b
2f **ret** $Q(x_2) \leftarrow y_2$

On right query $R_{P^{-1}}(y_1)$:

3a **if** $y_1 \in \mathrm{rng}(P)$ **ret** $x_1 = P^{-1}(y_1)$
3b $x_1 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{dom}(P)$
3c **if** $x_1 \in A_3$:
3d $\quad$ **bad**$_3 \leftarrow$ **true**
3e $\quad$ **GOTO** 3b
3f **ret** $P^{-1}(y_1) \leftarrow x_1$

On right query $R_{Q^{-1}}(y_2)$:

4a **if** $y_2 \in \mathrm{rng}(Q)$ **ret** $x_2 = Q^{-1}(y_2)$
4b $x_2 \xleftarrow{\$} \mathbb{Z}_2^l \backslash \mathrm{dom}(Q)$
4c **if** $x_2 \in A_4$:
4d $\quad$ **bad**$_4 \leftarrow$ **true**
4e $\quad$ **GOTO** 4b
4f **ret** $Q^{-1}(y_2) \leftarrow x_2$

On left query $L_3(M)$:

5a $(M'_1, \ldots, M'_k) \leftarrow \mathrm{pad}(M)$
5b $h_0 \leftarrow IV_n$
5c **for** $i = 1, \ldots, k$:
5d $\quad a \leftarrow R_Q(M'_i)$
5e $\quad b \leftarrow R_P(h_{i-1} \oplus M'_i)$
5f $\quad h_i \leftarrow a \oplus b \oplus h_{i-1}$
5g $d \leftarrow R_P(h_k)$
5h $h \leftarrow \mathrm{chop}_{l-n}(d \oplus h_k)$
5i **ret** h

$A_0 = \mathrm{rng}(P)$;

$A_1 = \bigcup_{(x_2,y_2)\in Q} \Big(\big\{ x_1 \oplus x_2 \oplus y_2 \oplus s \mid s \in V \cup (V_{\mathrm{new}} \backslash \{x_1 \oplus x_2 \oplus y_1 \oplus y_2\}) \big\} \cup \big\{ x_1 \oplus x_2 \oplus y_2 \oplus x'_1 \mid x'_1 \in \mathrm{dom}(P) \cup \{x_1\} \big\} \Big)$,
where $V_{\mathrm{new}} = \{x_1 \oplus x'_2, x_1 \oplus x'_2 \oplus y_1 \oplus y'_2 \mid (x'_2, y'_2) \in Q\}$ is a multiset;

$A_2 = \bigcup_{(x_1,y_1)\in P} \Big(\big\{ x_1 \oplus x_2 \oplus y_1 \oplus s \mid s \in V \cup (V_{\mathrm{new}} \backslash \{x_1 \oplus x_2 \oplus y_1 \oplus y_2\}) \big\} \cup \big\{ x_1 \oplus x_2 \oplus y_1 \oplus x'_1 \mid x'_1 \in \mathrm{dom}(P) \big\} \Big)$,
where $V_{\mathrm{new}} = \{x'_1 \oplus x_2, x'_1 \oplus x_2 \oplus y'_1 \oplus y_2 \mid (x'_1, y'_1) \in P\}$ is a multiset;

$A_3 = \bar{r}(V) \cup \{x_2 \oplus s \mid x_2 \in \mathrm{dom}(Q), s \in r(V)\}$;

$A_4 = \{x_1 \oplus s \mid x_1 \in \mathrm{dom}(P), s \in r(V)\}$.

Fig. 8. Game 7. The distinguisher has access to L_3^R, R.