Mix-Compress-Mix Revisited: Dispensing with Non-invertible Random Injection Oracles

Mohammad Reza Reyhanitabar and Willy Susilo

Centre for Computer and Information Security Research
School of Computer Science and Software Engineering
University of Wollongong, Australia
{rezar, wsusilo}@uow.edu.au

Abstract. We revisit the problem of building dual-model secure (DMS) hash functions that are simultaneously provably collision resistant (CR) in the standard model and provably pseudorandom oracle (PRO) in an idealized model. Designing a DMS hash function was first investigated by Ristenpart and Shrimpton (ASIACRYPT 2007); they put forth a generic approach, called Mix-Compress-Mix (MCM), and showed the feasibility of the MCM approach with a secure (but inefficient) construction. An improved construction was later presented by Lehmann and Tessaro (ASIACRYPT 2009). The proposed construction by Ristenpart and Shrimpton requires a *non-invertible* (pseudo-) random injection oracle (PRIO) and the Lehmann-Tessaro construction requires a *non-invertible* random permutation oracle (NIRP). Despite showing the feasibility of realizing PRIO and NIRP objects in theory–using ideal ciphers and (trapdoor) one-way permutations– these constructions suffer from several efficiency and implementation issues as pointed out by their designers and briefly reviewed in this paper.
In contrast to the previous constructions, we show that constructing a DMS hash function does not require any PRIO or NIRP, and hence there is no need for additional (trapdoor) one-way permutations. In fact, Ristenpart and Shrimpton posed the question of whether MCM is secure under easy-to-invert mixing steps as an open problem in their paper. We resolve this question in the affirmative in the fixed-input-length (FIL) hash setting. More precisely, we show that one can sandwich a provably CR function, which is *sufficiently compressing*, between two random *invertible* permutations to build a provably DMS compression function. Any multi-property-preserving (MPP) domain extender that preserves CR and PRO can then be used to convert such a DMS compression function to a full-fledged DMS hash function. Interestingly, there are *efficient* off-the-shelf candidates for all the three ingredients (provably CR compression functions, random invertible permutations, and MPP domain extenders) from which one can choose to implement such a DMS hash function in practice. Further, we also explain the implementation options as well as a concrete instantiation.

Key words: hash functions, provable security, collision resistance, pseudorandom oracle.

1 Introduction

There have been several attempts to construct provably secure hash functions in the standard model [15, 23, 11, 20, 7, 10]; however, these constructions usually guarantee only specific security properties (mainly the CR and one-way properties) and they are inappropriate candidates for real-world instantiation of random oracles, which renders them useless for many practical applications of hash functions [31, 26]. On the other hand, there are also provably secure hash functions in idealized models whose security, in the sense of the CR and one-way (OW) properties [9] or the PRO property [12, 5, 14, 6], is proven assuming that their underlying components are ideal objects (e.g. ideal ciphers or FIL random oracles), but outside these idealized models their actual security becomes unclear and unproven.

An interesting problem is how to construct a cryptographic hash function that has provable dual-model security; that is, both provably secure (e.g. in the sense of CR) in the standard model and provably PRO in an idealized model (e.g. the ideal cipher model).

Ristenpart and Shrimpton initiated an investigation of this problem in [26, 27]. Given a hash function H that is provably CR in the standard model and has some regularity properties (as defined in [26]), they showed how to construct a hash function F that inherits the provable CR property of H in the standard model while simultaneously being indifferentiable (in the sense of [21]) from an ideal hash function in an idealized model for the underlying components (i.e. assuming access to some finite idealized primitives such as an ideal cipher or a FIL random oracle). They presented a generic encapsulation method, called Mix-Compress-Mix (MCM), which sandwiches H between two injective mixing stages, $\mathcal{M}_1$ and $\mathcal{M}_2$, to get $F(.) = \mathcal{M}_2(H(\mathcal{M}_1(.)))$. It is proved that if the mixing stages are pseudorandom injection oracles (PRIO) and H is CR and possesses a suitable regularity property then F will be a pseudorandom oracle. A PRIO is defined in [26] as a pseudorandom oracle which observes injectivity but there is no associated inversion oracle; i.e., a PRIO is a "non-invertible" primitive by definition. Unfortunately, an efficient construction for instantiating a PRIO has turned out to be a non-trivial task. The Tag-and-Encryption (TE) construction was presented by Ristenpart and Shrimpton as a proof-of-concept, but it is inefficient and suffers from composability limitations as pointed out by the designers themselves. Furthermore, we note that the Ristenpart-Shrimpton construction (for F) needs three primitives: a hash function H (with the CR and regularity properties), a blockcipher E (to instantiate an ideal cipher), and an *additional* (complexity-theoretic) primitive; namely, a *trapdoor* one-way permutation.

Lehmann and Tessaro presented an improved MCM construction [19] resolving some of the problems of the Ristenpart-Shrimpton construction. However, the Lehmann-Tessaro construction to build F still needs three primitives: a hash function H, a blockcipher E, and an *additional* one-way permutation with some *special constraints* imposed on it. The need for an additional one-way permutation is due to the fact that the Lehmann-Tessaro construction still requires the output mixing stage ($\mathcal{M}_2$) to be a PRIO with zero stretch, which is called a "non-invertible" random permutation (NIRP) oracle in [19]. Let n be the hash size which is equal to the block size of E in the Lehmann-Tessaro construction. As discussed in [19], their proposed construction of NIPR requires a one-way permutation $P : \{0,1\}^n \rightarrow \{0,1\}^n$ that satisfies the following constraints: it must resist to inversion attacks with running time of roughly $O(2^{n/2})$ and its input/output length (i.e. n) must *equal* that of an existing block cipher.

As pointed out by Lehmann and Tessaro [19], one-way permutations on elliptic curves of prime orders [18] are the only known candidates to satisfy the first condition (security level of $O(2^{n/2})$), but their domain/range does not equal $\{0,1\}^n$; hence, they cannot directly be used in this construction (without possibly having to modify the whole construction and its proof of security). Therefore, a practical implementation for P (and hence $\mathcal{M}_2$) is left unclear and open at the conclusion of [19].

In addition to the aforementioned issues, we note an important limitation of indifferentiability guarantee of these designs; namely, in both of the previous constructions the indifferentiability is complexity-theoretic in nature, which is due to using an additional complexity-theoretic primitive by the constructions; namely, the former uses a trapdoor one-way permutation (in construction of TE) and the latter uses a one-way permutation (in construction of NIRP). Hence, *even if the starting hash function H is an ideal hash* (instead of a CR hash function in the standard model) the provided indifferentiabilty bounds of these constructions still remain complexity-theoretic due to relying on a computationally secure (trapdoor) one-way permutation.

We notice that the multi-property combiners of [16] can also provide a DMS hash construction, but the resulting construction doubles the output length of the underlying hash functions and is rather *inefficient* as also remarked in [19] (for example, the only combiner from [16] which can be used to construct a DMS hash function, called "$\mathrm{C}_{4\mathrm{P}\&\mathrm{IRO}}$", requires 8 calls to its underlying two hash functions plus a pairwise independent function).

Our Construction. Let $\pi_1 : \{0,1\}^m \rightarrow \{0,1\}^m$ and $\pi_2 : \{0,1\}^n \rightarrow \{0,1\}^n$ be two invertible permutations with associated inverses π_1^{-1} and π_2^{-1}, respectively. Let $H : \mathcal{K} \times \{0,1\}^m \rightarrow \{0,1\}^n$ be a FIL hash function. We show that the composition function $F = \pi_2 \circ H \circ \pi_1$ defined as $F : \mathcal{K} \times \{0,1\}^m \rightarrow \{0,1\}^n$ s.t. $F_K(M) = \pi_2(H_K(\pi_1(M)))$, for every $K \in \mathcal{K}$ and $M \in \{0,1\}^m$, has the following properties:

1. F is PRO if π_1 and π_2 are random invertible permutations and H is a CR and one-way hash function with suitable regularity properties.

2. F inherits all security properties of H in the standard model.

The second property above is straightforward to show noticing that π_1 and π_2 are easily invertible permutations (i.e., their inversion permutations are public); for example, these can be built using fixed-key block ciphers. To get a full-fledged variable-input-length (VIL) hash function $\mathcal{F} : \mathcal{K}' \times \{0,1\}^* \to \{0,1\}^{n'}$, one can easily extend the domain of our FIL hash function F using any existing efficient MPP domain extension transform [5, 6, 1, 2] that can preserve CR, PRO, and several other security notions of interest.

Compared to the previous two constructions, in our method: (1) there is no need for additional complexity-theoretic primitives, neither a trapdoor one-way permutation as in [26] nor a one-way permutation with special constraints yielding to practicality issues as in [19]), (2) indifferentiability can be information-theoretic if one uses a hash function H whose CR and OW properties are proved information-theoretically in an idealized model, such as any secure block cipher based hash functions in [28, 9]. The latter is actually a corollary of the former.

We note that if H is a sufficiently compressing function (e.g. $m \geq 2n$) then the OW (preimage resistance) property is actually implied by the CR property [25], albeit up to the birthday bound (Section 2 provides some details). The PRO proof for our scheme as well as the PRO proofs of Ristenpart-Shrimpton [27] and Lehmann-Tessaro [19] schemes are only proving security up to the birthday bound.

Implementation. For an efficient implementation of a DMS hash function according to our proposed method, one needs three components as follows:

1. An *efficient* provably CR function with suitable regularity properties, e.g. SWIFFT [20] or VSH* [7].
2. Two *efficient* candidates to instantiate the random invertible permutations for the input and output mixing stages. There are several efficient and widely-evaluated dedicated designs for random invertible permutations of different (large) sizes in the literature; for example, the class of permutations E_d in the JH hash function [32] (which are based on the d-dimensional generalized AES design methodology), the Keccak-f permutations of the Keccak hash function [8], or the P and Q permutations of the Grøstl hash function [17]. (JH, Keccak and Grøstl are among the five final-round SHA-3 candidates [22]. Their indifferentiability proofs assume that these underlying permutations are random.)
3. An efficient MPP domain extender that preserves CR and PRO, e.g. HAIFA [2]. If preservation of other properties in addition to CR and PRO are also aimed then more powerful MPP transforms [6, 1] should be be used as HAIFA does not preserve some properties [1].

As a concrete implementation example, one can use SWIFFT with parameters $m = 1024$ bits and $n = 512$ bits as the underlying provably CR function, and the 512-bit permutation P_{512} and the 1024-bit permutation Q_{1024} from the Grøstl compression function [17] as the mixing stages. This yields to a DMS compression function with input length of 1024 bits and output length of 512 bits. The HAIFA construction can then be applied to obtain a full-fledged DMS hash function. It is worth noticing that provably secure hash functions such as SWIFFT (using a reduction from an underlying hard problem) usually do not provide an ideally expected level of concrete security with respect to the CR and OW properties; i.e., for hash size of n bits the expected levels of CR and OW security provided by these functions are usually (much) less than the ideal levels of $2^{\frac{n}{2}}$ and 2^n (for the CR and OW properties, respectively). This is similar to the situation for public key primitives like RSA where one needs to estimate concrete security levels for specific parameter settings based on the best known algorithms to solve the underlying hard problem. For example, the currently known algorithms to find collisions and preimages in the SWIFFT function with a 512-bit hash size have time complexities 2^{106} and 2^{448}, respectively [20].

Reset Indifferentiability. Our treatment of PROs in this paper is based on the formalization of Coron et al. in [12] following the original indifferentiability framework of Maurer et al. in [21]. Ristenpart et al. in Eurocrypt 2011 [29] showed limitations of the indifferentiability composition theorem when applying it to a general cryptosystem, requiring a security notion that is defined by games involving multiple, disjoint adversarial stages. They put forth "reset indifferentiability" as a new stronger notion to handle this issue; however, to the best of our knowledge it is still an open problem how to design a hash construct (*even an inefficient one*) that can satisfy this new stronger notion. As shown in [29] practical (single-pass) hash functions are not reset indifferentiable.

Organization of the Paper. Section 2 provides the required preliminaries and conventions used throughout the paper. Formal description of the construction and its security analysis are provided in Section 3. Section 4 and appendices contain the proofs.

2 Preliminaries

Notations and Conventions. If S is a finite set, $x \overset{\$}{\leftarrow} S$ means that x is chosen from S uniformly at random; $|S|$ denotes the size of S. $X \leftarrow Y$ is used for denoting a normal assignment statement where the value of Y is assigned to X. The set of all binary strings of length n bits (for some positive integer n) is denoted as $\{0,1\}^n$, the set of all binary strings whose lengths are variable but upper-bounded by N is denoted by $\{0,1\}^{\leq N}$ and the set of all binary strings of arbitrary length is denoted by $\{0,1\}^*$. The symbol $\perp$ means that the value of a variable is yet undefined, $\wedge$ denotes logical 'AND' operation, and $\vee$ denotes logical 'OR' operation. If S_1 and S_2 are two sets we denote their union by $S_1 \cup S_2$ and their subtraction by either $S_1 \backslash S_2$ or $S_1 - S_2$. By $i, j < k$ we mean $(i < k) \wedge (j < k)$. Let A be an adversary that returns a binary value; by $A^{f(.)}(X) \Rightarrow 1$ we refer to the event that the adversary A with input X and access to oracle $f(.)$ returns value 1. By time complexity of an algorithm we mean the running time, relative to some fixed model of computation plus the size of the description of the algorithm using some fixed encoding method. We denote the set of all functions with domain $\{0,1\}^m$ and range $\{0,1\}^n$ by $\mathrm{Func}(m,n)$ and the set of all permutations over $\{0,1\}^m$ by $\mathrm{Perm}(m)$.

We denote a FIL hash function (or compression function) by $H : \mathcal{K} \times \{0,1\}^m \rightarrow \{0,1\}^n$, where m and n are two positive integers such that $n < m$, and the keyspace $\mathcal{K}$ is a non-empty set of strings. By convention if $|\mathcal{K}| = 1$ we assume that $\mathcal{K} = \{\varepsilon\}$; i.e., it only consists of the empty string, and in this case we call H a keyless compression function (which can be simply denoted as a one-argument function $H : \{0,1\}^m \rightarrow \{0,1\}^n$). If $|\mathcal{K}| \geq 2$ we call H a compression function family or a dedicated-key compression function. We use the notations $H_K(M)$ and $H(K, M)$ interchangeably. Time_H denotes the time complexity of computing $H_K(X)$ for any $K \in \mathcal{K}$ and $X \in \{0,1\}^m$, plus the time complexity for sampling from $\mathcal{K}$).

As usual in concrete-security definitions, the resource parameterized function $\mathbf{Adv}_H^{\mathsf{xxx}}(\mathbf{r})$ denotes the maximal value of the adversarial advantage (*i.e.* $\mathbf{Adv}_H^{\mathsf{xxx}}(\mathbf{r}) = max_A \{\mathbf{Adv}_H^{\mathsf{xxx}}(A)\}$) over all adversaries A, against the xxx property of H, that use resources bounded by $\mathbf{r}$. The resource parameter $\mathbf{r}$, depending on the notion, may include time complexity (t), length of queries and number of queries that an adversary makes to its oracles (if any).

CR and OW Properties. Let $H : \mathcal{K} \times \{0,1\}^m \rightarrow \{0,1\}^n$ be a compression function. The advantage measures for an adversary A against the CR and OW (or Preimage Resistance) properties are defined as follows:

- $\mathbf{Adv}_H^{CR}(A) = \Pr\left[K \overset{\$}{\leftarrow} \mathcal{K}; (M, M') \overset{\$}{\leftarrow} A(K) : M \neq M' \ \wedge \ H_K(M) = H_K(M')\right]$
- $\mathbf{Adv}_H^{OW}(A) = \Pr\left[K \overset{\$}{\leftarrow} \mathcal{K}; M \overset{\$}{\leftarrow} \{0,1\}^m; Y \leftarrow H_K(M); M' \overset{\$}{\leftarrow} A(K, Y) : H_K(M') = Y\right]$

CR provisionally implies **OW**. From [25] we have $\mathbf{Adv}_H^{OW}(t) \leq 2\mathbf{Adv}_H^{CR}(t') + 2^{n-m}$, where $t' = t + c\mathrm{Time}_H$ for some small constant c. This is called a "provisional implication" [25] where the strength of the implication depends on the amount of compression achieved by the hash function (due to the 2^{n-m} term in the bound). If the hash function is substantially compressing, e.g., mapping $2n$ bits to n bits, then the implication is a strong one (i.e. $\mathbf{Adv}_H^{OW}(t) \leq 2\mathbf{Adv}_H^{CR}(t') + 2^{-n}$). That is, a sufficiently compressing CR function implies an OW function, albeit up to the birthday bound.

Keyless Hash Functions. Our results can be straightforwardly adapted to keyless hash functions, using the human-ignorance framework of Rogaway [24] when dealing with the CR property for keyless hash functions.

PRO. The indifferentiability framework [21] captures the definitions for comparing a given object F that utilizes some public components (e.g. fixed-input-length random oracles or random permutations) in its construction with an idealized object $\mathcal{R}$. Let $F^{f_1,\cdots,f_\ell} : \mathcal{K} \times \mathit{Dom} \rightarrow \mathit{Rng}$ be a function family that has access to public oracles $f_1, \cdots, f_\ell$. For the purpose of the PRO property [12, 5, 6] the idealized function to which we compare a member function $F_K^{f_1,\cdots,f_\ell} : \mathit{Dom} \rightarrow \mathit{Rng}$ is a random oracle $\mathcal{R} : \mathit{Dom} \rightarrow \mathit{Rng}$. Let $S^{\mathcal{R}} = (S_1, \cdots, S_\ell)$ be a simulator that has oracle access to $\mathcal{R}$ and exposes interfaces for each of the ℓ oracles used by F. The aim of the simulator is to mimic the oracles $f_1, \cdots, f_\ell$ such that no adversary can tell apart whether it is interacting with the construction F and oracles $(f_1, \cdots, f_\ell)$ or with the ideal function $\mathcal{R}$ and the simulator's subroutines $(S_1, \cdots, S_\ell)$. Note that the simulator does not get to see adversary's queries to the ideal function $\mathcal{R}$. It is assumed that the key K for the hash function is given to the simulator as a parameter. The PRO advantage of an adversary A is defined as

$$\mathbf{Adv}_{F,\,S}^{pro}(A) = \left|\Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{F_K^{f_1,\cdots,f_\ell},\, f_1,\cdots,f_\ell}(K) \Rightarrow 1\right] - \Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{R},\, S^{\mathcal{R}}(K)}(K) \Rightarrow 1\right]\right|.$$

Regarding the resources, we measure the total number of queries that A makes to its $(\ell + 1)$ oracles. We also specify the resources utilized by S, namely, the total number of queries q_S made by S to $\mathcal{R}$ and the maximum running time t_S. (The values of the simulator's resources are generally functions of the number of queries made by an adversary.)

Regularity and Hash Function Balance. Bellare and Kohno [3] introduced a measure of the "amount of regularity" of a hash function (both keyless and dedicated-keyed ones) called "balance". They showed how the success probability of the birthday attack for finding collisions under a hash function H depends on the hash function balance as well as the size of the range of the hash function and the number of trials. (In the birthday attack to find collisions for a hash function, adversary simply picks q random points from the domain of the hash function and computes their hash values hoping that a pair of input points will have the same hash value.) Let $C_H(q)$ be the probability that the birthday attack on hash function H succeeds in finding a collision in q trials.

In this paper, we only need the definitions for the case of FIL hash functions. First, let's consider a keyless compression function $H : \{0,1\}^m \rightarrow \{0,1\}^n$, where $m > n \geq 1$. Let $r = 2^n$ and $d = 2^m$. For $i = 1, \cdots, r$ let $H^{-1}(Y_i)$ be the set of all preimages of Y_i under H; that is, the set of all $M \in \{0,1\}^m$ such that $H(M) = Y_i$, and let $d_i = |H^{-1}(Y_i)|$ be the size of this set. The balance of H is defined as

$$\mu(H) = \log\left[\frac{d^2}{d_1^2 + \cdots + d_r^2}\right]$$

where $\log_r(.)$ denote the logarithm in base r. The following results are from [3]:

- $0 \leq \mu(H) \leq 1$; the maximum balance of 1 is achieved when the hash function H is regular (i.e. we have $d_i = d/r$ for all i), while the minimum balance of 0 is achieved when H is a constant function.
- $C_H(q) \leq \frac{q(q-1)}{2}) \left[\frac{1}{r^{\mu(H)}} - \frac{1}{d}\right]$, or approximately we have $C_H(q) \leq \frac{0.5q^2}{r^{\mu(H)}}$ when $d \geq 2r \geq 4$.

The generalization of the balance measure and related results for the case of a dedicated-key hash function are also provided in [3]. Let $H : \mathcal{K} \times \{0,1\}^m \rightarrow \{0,1\}^n$ be a dedicated-key hash function. For each fixed

$K \in \mathcal{K}$, let $H_K : \{0,1\}^m \rightarrow \{0,1\}^n$ be defined as $H_K(.) = H(K,.)$. The definitions for the metric $C_H(q)$ and the balance measure $\mu(H)$ for this setting of dedicated-keyed hash function (hash function family) are given in [3] as $C_H(q) = \frac{1}{|\mathcal{K}|} \cdot \sum_{K \in \mathcal{K}} C_{H_K}(q)$ and $\mu(H) = \log_r \left[\frac{1}{|\mathcal{K}|} \cdot \sum_{K \in \mathcal{K}} \frac{1}{r^{\mu(H_K)}}\right]^{-1}$; their relation is given by

$$C_H(q) \leq \frac{q(q-1)}{2}\left[\frac{1}{r^{\mu(H)}} - \frac{1}{d}\right] \approx \frac{0.5q^2}{r^{\mu(H)}}$$

We will also use the following two notions of regularity in our proofs:

ϵ-almost output regularity. A function $H : \mathcal{K} \times \{0,1\}^m \rightarrow \{0,1\}^n$ is ϵ-almost output regular if for any adversary A: $|\Pr[K \overset{\$}{\leftarrow} \mathcal{K}, M \overset{\$}{\leftarrow} \{0,1\}^m ; Y \leftarrow H_K(M) : A(K,Y) \Rightarrow 1] - \Pr[K \overset{\$}{\leftarrow} \mathcal{K}, Y \overset{\$}{\leftarrow} \{0,1\}^n : A(K,Y) \Rightarrow 1]| \leq \epsilon$.

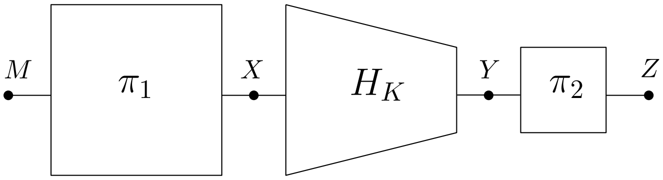
Δ-regularity [26]. For a function $H : \mathcal{K} \times \{0,1\}^m \rightarrow \{0,1\}^n$, let $\delta(K,Y) = \left|\frac{|H_K^{-1}(Y)| - 2^{m-n}}{2^m}\right|$ and $\Delta_K = \max\left(\delta(K,Y)\right)$, where the maximum is taken over all values of Y. We say that H is Δ-regular if $\sum_{K \in \mathcal{K}} p_K \Delta_K \leq \Delta$, where $p_K = \Pr\left[K = K' : K' \overset{\$}{\leftarrow} \mathcal{K}\right]$. This is a measure of the average (over keys) maximum deviation from regularity.

Game-Playing Technique [30, 4]. We use the code-based game-playing framework of [4] in our proof. A game G is a program (written in pseudocode) that consists of an initialization procedure (Initialize(.)), a finalization procedure (Finalize(.)), and oracle procedures $P_1(.), P_2(.), \cdots, P_n(.)$ for some $n \geq 1$. Adversary can make calls to the oracle procedures passing in parameters from some finite domain associated to each oracle. To run game $G = (\text{Initialize}, P_1, P_2, \cdots, P_n, \text{Finalize})$ with adversary A, first procedure Initialize is called with an input string parameter *param* (in our proof this is an empty string). Then we run A, passing it any value that was returned by Initialize. When A calls its i-th oracle P_i with a string, we pass that string to P_i and return to the adversary whatever P_i returns. When A finally halts with some output *out*, we pass *out* to Finalize which generates an output for the game. When the output of the game is the same as the output of the adversary we delete Finalize. We write $\Pr[A^G \Rightarrow 1]$ for the probability that the adversary A outputs 1 when G is run with A. The notation $\Pr[G^A \Rightarrow 1]$ denotes the probability that the output of game G (i.e. output of its Finalize procedure) is 1 when G is run with A. If there is no Finalize then $\Pr[A^G \Rightarrow 1] = \Pr[G^A \Rightarrow 1]$. The advantage of A in distinguishing two games G and H is defined as $\mathbf{Adv}(A^G, A^H) = \left|\Pr[A^G \Rightarrow 1] - \Pr[A^H \Rightarrow 1]\right|$. For any three games G, I and H, we have the triangle inequality $\mathbf{Adv}(A^G, A^H) \leq \mathbf{Adv}(A^G, A^I) + \mathbf{Adv}(A^I, A^H)$ which is used to bound the adversarial advantage when a sequence of games is used during the proof. We refer to [4] for further conventions used in the code-based game-playing framework.

Pointless Queries. We assume that an adversary A does not make redundant (or pointless) queries to its oracles: (1) a query is redundant if it has been made before; (2) given a permutation oracle $\mathbf{\Pi}(.)$ and its inverse oracle $\mathbf{\Pi^{-1}}(.)$, a query $\mathbf{\Pi}(X)$ is redundant if A has previously received X in answer to a query $\mathbf{\Pi^{-1}}(Y)$; a query $\mathbf{\Pi^{-1}}(Y)$ is redundant if A has previously received Y in answer to a query $\mathbf{\Pi}(X)$. Disallowing redundant queries is clearly without loss of generality in the sense that from any arbitrary adversary A that makes q queries, one can make an adversary B that asks at most q non-redundant queries and achieves the same advantage as A.

3 Construction Description and Security Analysis

Construction Description. Fig. 1 illustrates our proposed FIL MCM function $F : \mathcal{K} \times \{0,1\}^m \rightarrow \{0,1\}^n$, defined as $F_K(M) = \pi_2(H_K(\pi_1(M)))$, for $M \in \{0,1\}^m$ and $K \in \mathcal{K}$, where $\pi_1 : \{0,1\}^m \rightarrow \{0,1\}^m$ and



```
Game $G_0$

Procedure Initialize
    $\pi_1 \xleftarrow{\$} \mathrm{Perm}(m)$
    $\pi_2 \xleftarrow{\$} \mathrm{Perm}(n)$
    return $K \xleftarrow{\$} \mathcal{K}$

Procedure $\mathcal{O}_0(M)$
    $X \leftarrow \pi_1[M]$
    $Y \leftarrow H_K(X)$
    return $Z \leftarrow \pi_2[Y]$

Procedure $\mathcal{O}_1(M)$
    return $X \leftarrow \pi_1[M]$

Procedure $\mathcal{O}_1^{-1}(X)$
    return $M \leftarrow \pi_1^{-1}[X]$

Procedure $\mathcal{O}_2(Y)$
    return $Z \leftarrow \pi_2[Y]$

Procedure $\mathcal{O}_2^{-1}(Z)$
    return $Y \leftarrow \pi_2^{-1}[Z]$
```

Fig. 1. (Left) FIL MCM construction F using easily invertible permutations as the mixing stages. **(Right)** Game G_0, which is used in the indifferentiability proof, captures the behavior of the real setting where an adversary A has access to the following five oracles: $\mathcal{O}_0$ which realizes construction F, $\mathcal{O}_1$ and $\mathcal{O}_2$ which realize two random permutations, and oracles $\mathcal{O}_1^{-1}$ and $\mathcal{O}_2^{-1}$ which realize the inverses of the random permutations, respectively.

$\pi_2 : \{0,1\}^n \rightarrow \{0,1\}^n$ are two permutations with given inverses π_1^{-1} and π_2^{-1}, respectively. Game G_0 in Fig. 1 describes the oracles which are provided in this real setting for a differentiating adversary.

Simulator Description. Let A be an adversary that wants to differentiate our FIL MCM function $F_K : \{0,1\}^m \rightarrow \{0,1\}^n$ from a truly random function $\mathcal{R} : \{0,1\}^m \rightarrow \{0,1\}^n$. We remind that the output of the **Initialize** procedure, i.e. the key K, is given as an input to the adversary and the simulator. As shown in Fig. 1, in the real setting when A interacts with construction F and its public (permutation) components, it is provided with five oracles: oracle $\mathcal{O}_0$ which realizes F, oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ which realize two random permutations, and oracles $\mathcal{O}_1^{-1}$ and $\mathcal{O}_2^{-1}$ which realize the inverses of the random permutations, respectively. Figure 2 shows Game I_0, capturing the behavior of the simulated setting, where oracles $\mathcal{O}_1$, $\mathcal{O}_1^{-1}$, $\mathcal{O}_2$ and $\mathcal{O}_2^{-1}$ are implemented by a simulator $\mathcal{S} = (\mathcal{S}_{\pi_1}, \mathcal{S}_{\pi_1^{-1}}, \mathcal{S}_{\pi_2}, \mathcal{S}_{\pi_2^{-1}})$. The adversary has direct oracle access to $\mathcal{O}_0$ and the simulator does not get to see the adversary's query-response pairs to this oracle.

The simulator keeps a memory of all previously answered queries and their associated variables in a set $\mathcal{C}$ of commitments. Each member of $\mathcal{C}$ is a tuple (M, X, Y, Z) that holds the corresponding values of the variables in the construction of F as shown in Fig. 1; if a value is yet unknown (not defined yet) it is denoted by $\perp$. On each query, not only $\mathcal{S}$ chooses its answer, but also it chooses and stores values of the associated variables according to the construction of F when this is possible; otherwise, it stores a $\perp$ for the value of a variable that cannot be determined appropriately yet. We describe subroutines of $\mathcal{S}$ as shown in Fig. 2 in the following. We start by explaining $\mathcal{S}_{\pi_2}$ (for answering $\mathcal{O}_2(Y)$ queries) and $\mathcal{S}_{\pi_2^{-1}}$ (for answering $\mathcal{O}_2^{-1}(Z)$ queries), as these are the queries that possibly can cause difficulties for the simulator when later answering some related $\mathcal{O}_1(M)$ and $\mathcal{O}_1^{-1}(X)$ queries (the difficulty lies in the fact that the simulator can neither invert the hash function H nor the random function $\mathcal{R}$).

On query $\mathcal{O}_2(Y)$, subroutine $\mathcal{S}_{\pi_2}$ checks the memory $\mathcal{C}$. If a commitment Z has already been made specifying how to answer this query Y, it is returned as the answer (at line 040); otherwise, a random M is chosen and the value $Z \leftarrow \mathcal{R}[M]$ is returned as the answer (but M is not revealed to the adversary). The simulator also does the following two housekeeping actions: set $\mathcal{C}$ is updated to include tuple $(M, \perp, Y, Z)$ (note that the value of X cannot be determined without finding a corresponding preimage of Y under H which is assumed to be hard; hence, it is left unknown at this point), and the random value M is stored in a set $\mathcal{P}_e$ which is used for recording the ("poisoned") queries that if are asked later by the adversary, in a query $\mathcal{O}_1(M)$, can cause the simulator to fail (**return** $\perp$ at line 020).

Game I_0

```
Procedure Initialize
$\mathcal{R} \xleftarrow{\$} \mathrm{Func}(m, n)$
return $K \xleftarrow{\$} \mathcal{K}$

Procedure $\mathcal{O}_1(M)$
if $M \in \mathcal{P}_e$ then $bad \leftarrow$ true, return $\perp$
$Z \leftarrow \mathcal{R}[M]$
if $\exists\, (\perp, X, Y, Z) \in \mathcal{C}$ then
    $\mathcal{C} \leftarrow (\mathcal{C} \backslash \{(\perp, X, Y, Z)\}) \cup \{(M, X, Y, Z)\}$
    $\mathcal{P}_d \leftarrow \mathcal{P}_d \backslash \{X\}$, return $X$
$X \xleftarrow{\$} \{0,1\}^m$
$Y \leftarrow H_K(X)$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(M, X, Y, Z)\}$
return $X$

Procedure $\mathcal{O}_2(Y)$
if $\exists\, (M, X, Y, Z) \in \mathcal{C}$ then return $Z$
$M \xleftarrow{\$} \{0,1\}^m$
$Z \leftarrow \mathcal{R}[M]$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(M, \perp, Y, Z)\}$
$\mathcal{P}_e \leftarrow \mathcal{P}_e \cup \{M\}$
return $Z$
```

```
Procedure $\mathcal{O}_0(M)$
return $Z \leftarrow \mathcal{R}[M]$

Procedure $\mathcal{O}_1^{-1}(X)$
if $X \in \mathcal{P}_d$ then $bad \leftarrow$ true, return $\perp$
$Y \leftarrow H_K(X)$
if $\exists\, (M', X', Y, Z') \in \mathcal{C} \wedge X \neq X'$ then $bad \leftarrow$ true, return $\perp$
if $\exists\, (M, \perp, Y, Z) \in \mathcal{C}$ then
    $\mathcal{C} \leftarrow (\mathcal{C} \backslash \{(M, \perp, Y, Z)\}) \cup \{(M, X, Y, Z)\}$
    $\mathcal{P}_e \leftarrow \mathcal{P}_e \backslash \{M\}$, return $M$
$M \xleftarrow{\$} \{0,1\}^m$
$Z \leftarrow \mathcal{R}[M]$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(M, X, Y, Z)\}$
return $M$

Procedure $\mathcal{O}_2^{-1}(Z)$
if $\exists\, (M, X, Y, Z) \in \mathcal{C}$ then return $Y$
$X \xleftarrow{\$} \{0,1\}^m$
$Y \leftarrow H_K(X)$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(\perp, X, Y, Z)\}$
$\mathcal{P}_d \leftarrow \mathcal{P}_d \cup \{X\}$
return $Y$
```

Game I_1

```
Procedure Initialize
$\mathcal{R} \xleftarrow{\$} \mathrm{Func}(m, n)$
return $K \xleftarrow{\$} \mathcal{K}$

Procedure $\mathcal{O}_1(M)$
if $M \in \mathcal{P}_e$ then $bad \leftarrow$ true, return $\perp$
$Z \leftarrow \mathcal{R}[M]$
if $\exists\, (\perp, X, Y, Z) \in \mathcal{C}$ then
    $\mathcal{C} \leftarrow (\mathcal{C} \backslash \{(\perp, X, Y, Z)\}) \cup \{(M, X, Y, Z)\}$
    $\mathcal{P}_d \leftarrow \mathcal{P}_d \backslash \{X\}$, return $X$
$X \xleftarrow{\$} \{0,1\}^m$
$Y \leftarrow H_K(X)$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(M, X, Y, Z)\}$
return $X$

Procedure $\mathcal{O}_2(Y)$
$M \xleftarrow{\$} \{0,1\}^m$
$Z \leftarrow \mathcal{R}[M]$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(M, \perp, Y, Z)\}$
$\mathcal{P}_e \leftarrow \mathcal{P}_e \cup \{M\}$
return $Z$
```

```
Procedure $\mathcal{O}_0(M)$
return $Z \leftarrow \mathcal{R}[M]$

Procedure $\mathcal{O}_1^{-1}(X)$
if $X \in \mathcal{P}_d$ then $bad \leftarrow$ true, return $\perp$
$Y \leftarrow H_K(X)$
if $\exists\, (M', X', Y, Z') \in \mathcal{C} \wedge X \neq X'$ then $bad \leftarrow$ true, return $\perp$
if $\exists\, (M, \perp, Y, Z) \in \mathcal{C}$ then
    $\mathcal{C} \leftarrow (\mathcal{C} \backslash \{(M, \perp, Y, Z)\}) \cup \{(M, X, Y, Z)\}$
    $\mathcal{P}_e \leftarrow \mathcal{P}_e \backslash \{M\}$, return $M$
$M \xleftarrow{\$} \{0,1\}^m$
$Z \leftarrow \mathcal{R}[M]$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(M, X, Y, Z)\}$
return $M$

Procedure $\mathcal{O}_2^{-1}(Z)$
$X \xleftarrow{\$} \{0,1\}^m$
$Y \leftarrow H_K(X)$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(\perp, X, Y, Z)\}$
$\mathcal{P}_d \leftarrow \mathcal{P}_d \cup \{X\}$
return $Y$
```

Fig. 2. (Top) Game I_0 captures the behavior of the simulated setting, where an adversary A has access to the following five oracles: oracle $\mathcal{O}_0$ which realize a random function, oracles $\mathcal{O}_1$, $\mathcal{O}_1^{-1}$, $\mathcal{O}_2$ and $\mathcal{O}_2^{-1}$ are implemented by the simulator $\mathcal{S}$ in order to make adversary A unable to differentiable this setting from the real setting. **(Bottom)** Game I_1 captures the behavior of the simplified simulated setting considering a simplified adversary $\mathcal{D}$, where oracles $\mathcal{O}_1, \mathcal{O}_1^{-1}, \mathcal{O}_2$ and $\mathcal{O}_2^{-1}$ are implemented by the simplified simulator $\mathcal{S}'$.

On query $\mathcal{O}_2^{-1}(Z)$, subroutine $\mathcal{S}_{\pi_2^{-1}}$ checks the memory $\mathcal{C}$. If a commitment Y has already been made specifying how to answer this query Z, it is returned as the answer (at line 050); otherwise, a random X is chosen and the value $Y \leftarrow H_K(X)$ is returned as the answer. We note that X is not revealed directly to the adversary; the only information that adversary gets about X is via the returned hash value $H(X)$. The simulator also performs the following two actions: set $\mathcal{C}$ is updated to include the tuple $(\perp, X, Y, Z)$ (note that the value of M cannot be determined as there is no inversion oracle for $\mathcal{R}$; hence, it is left unknown at this stage), and the random value X is stored in a set $\mathcal{P}_d$ which is used for recording the (poisoned) queries that if are asked later by the adversary, in a query $\mathcal{O}_1^{-1}(X)$, can cause the simulator to fail (**return** $\perp$ at line 030).

On query $\mathcal{O}_1(M)$, subroutine $\mathcal{S}_{\pi_1}$ first checks whether $M \in \mathcal{P}_e$, i.e. whether it is a (poisoned) query for which the corresponding value of X was left unknown in an earlier point at line 043. If this is the case then the simulator fails and aborts (at line 020). Otherwise, it queries $\mathcal{R}[.]$ on input M to get $Z \leftarrow \mathcal{R}[M]$. Now there are two cases. If the condition at line 022 is true, meaning that M may be linked to an existing tuple in the commitment set $\mathcal{C}$, then the corresponding value for X is returned (line 024), while updating the commitment set accordingly (line 023) and omitting X from the set of (poisoned) queries $\mathcal{P}_d$. Note that by the assumption that pointless queries are disallowed, such a returned value X from $\mathcal{O}_1(M)$ cannot actually be asked later in a query $\mathcal{O}_1^{-1}(X)$. If the condition at line 022 is not true, then the simulator returns a random value X (lines 025 and 028), and also computes $Y = H_K(X)$ and stores the complete tuple (M, X, Y, Z) in its commitment set $\mathcal{C}$.

Description of $\mathcal{S}_{\pi_1^{-1}}$, answering $\mathcal{O}_1^{-1}(.)$ queries in Game I_0, is very similar to that of $\mathcal{S}_{\pi_1}$ in most parts. We only note that there is an additional condition at line 032 which can make the simulator fail in this case (**return** $\perp$ at line 032), and that is if the adversary can make a collision happen under the hash function H.

Results. We are now ready to state our main result about the indifferentiability of our proposed FIL MCM construction with invertible mixing stages.

Theorem 1 (Main Theorem). *Let $\pi_1 : \{0,1\}^m \to \{0,1\}^m$ and $\pi_2 : \{0,1\}^n \to \{0,1\}^n$ be two random permutations with given associated inverses π_1^{-1} and π_2^{-1}, respectively. Let $H : \mathcal{K} \times \{0,1\}^m \to \{0,1\}^n$ be a Δ-regular and ϵ-almost output regular FIL hash function with balance value $\mu(H)$. Let $F : \mathcal{K} \times \{0,1\}^m \to \{0,1\}^n$ be the composed function defined by $F_K(M) = \pi_2(H_K(\pi_1(M)))$, for every $M \in \{0,1\}^m$ and $K \in \mathcal{K}$. Let A be an adversary that runs in time t and makes at most $(q_0, q_1, q_{-1}, q_2, q_{-2})$ queries to its five oracles $(\mathcal{O}_0, \mathcal{O}_1, \mathcal{O}_1^{-1}, \mathcal{O}_2, \mathcal{O}_2^{-1})$, respectively; let $q = q_0 + q_1 + q_{-1} + q_2 + q_{-2}$ be the total number of queries. Let $\mathcal{S}$ be the simulator that implements oracles $\mathcal{O}_1, \mathcal{O}_1^{-1}, \mathcal{O}_2$ and $\mathcal{O}_2^{-1}$ for the adversary A as shown in Game I_0 in Fig. 2. There exist adversaries B and C such that*

$$\mathbf{Adv}_{F,\mathcal{S}}^{pro}(A) \leq \mathbf{Adv}_H^{CR}(B) + q\mathbf{Adv}_H^{OW}(C) + \epsilon + \frac{q}{2^m} + q^2\left(\frac{4}{2^m} + \frac{3.5}{2^n} + \frac{1.5}{2^{n\mu(H)}} + 2\Delta\right)$$

where $\mathcal{S}$ runs in time $t_{\mathcal{S}} \leq c\,((q_1 + q_{-1} + q_{-2})\mathrm{Time}_H + q\log q)$ and makes $(q_1 + q_{-1} + q_2)$ oracle queries. Adversary B runs in time at most $t_B \leq t + c\,(q\mathrm{Time}_H + q\log q)$ and adversary C runs in time $t_C \leq t + t_{\mathcal{S}}$. □

4 Proof of Theorem 1

Overview. We use the game-playing technique to bound $\mathbf{Adv}_{F,\mathcal{S}}^{\mathrm{pro}}(A) = \mathbf{Adv}(A^{G_0},\ A^{I_0})$. The proof is divided into four lemmas. First, we provide the lemmas and the intuition behind their statements to conclude the proof of Theorem 1; then, we proceeded to prove the lemmas.

Lemma 1 shows that, without loss of generality, we can simplify the simulator $\mathcal{S}$ (underlying oracles $\mathcal{O}_1, \mathcal{O}_1^{-1}, \mathcal{O}_2$ and $\mathcal{O}_2^{-1}$ in Game I_0) provided that we only consider a certain class of simplified adversaries.

Namely, we show that the problem of bounding $\mathbf{Adv}(A^{G_0}, A^{I_0})$, where A is an arbitrary adversary and $\mathcal{S}$ is the original simulator (in Game I_0), can be reduced to bounding $\mathbf{Adv}(\mathcal{D}^{G_0}, \mathcal{D}^{I_1})$, where $\mathcal{D}$ is a simplified adversary and $\mathcal{S}'$ is the simplified simulator that implements oracles $\mathcal{O}_1, \mathcal{O}_1^{-1}, \mathcal{O}_2$ and $\mathcal{O}_2^{-1}$ in Game I_1, shown in Fig. 2. By simplified adversary we mean an adversary which does not ask some specific sequences of queries, but otherwise is arbitrary. Informally speaking, the assumption that adversary is simplified, in turn, allows us to simplify the simulator by omitting parts of its code which are responsible for taking care of those specific sequence of queries. We formally define a simplified adversary in the context of our proof in this paper in Definition 1.

To bound $\mathbf{Adv}(\mathcal{D}^{G_0}, \mathcal{D}^{I_1})$, we first specify two sequences of games to move G_0 and I_1 closer to each other. Namely, in Lemma 2 we specify a sequence of games $I_1 \rightarrow I_2 \rightarrow I_3$ (shown in Fig. 4) and bound $\mathbf{Adv}(\mathcal{D}^{I_1}, \mathcal{D}^{I_3})$, and in Lemma 3 we specify a sequence of games $G_0 \rightarrow G_1 \rightarrow G_2 \rightarrow G_3 \rightarrow G_4 \rightarrow G_5$ (shown in Fig. 1, Fig. 5 and Fig. 6) and bound $\mathbf{Adv}(\mathcal{D}^{G_0}, \mathcal{D}^{G_5})$.

Finally, in Lemma 4 we bound $\mathbf{Adv}(\mathcal{D}^{G_5}, \mathcal{D}^{I_3})$. (The proof of Lemma 4 itself includes further sequences of games.) The proof of Theorem 1 is then concluded combining the results of these lemmas.

Definition 1 (Simplified Adversary). *Let $\mathcal{D}$ be a pro adversary against F that makes at most q queries to all of its five oracles $\mathcal{O}_0, \mathcal{O}_1, \mathcal{O}_1^{-1}, \mathcal{O}_2$ and $\mathcal{O}_2^{-1}$. Let* `history` $= \{(tw_i, \alpha_i, \beta_i)\}$ *denote $\mathcal{D}$'s query/response transcript where $1 \leq i \leq q$ and $tw_i \in \{0, +1, -1, +2, -2\}$ specifies the oracle to which the i-th query was made; i.e., $tw_i = 0$ specifies oracle $\mathcal{O}_0$, and $tw_i = +1, -1, +2$ and -2 specify $\mathcal{O}_1, \mathcal{O}_1^{-1}, \mathcal{O}_2$ and $\mathcal{O}_2^{-1}$, respectively. (α_i, β_i) denotes the i-th (query, response) pair when $tw_i = 0, +1, +2$ or (response, query) pair when $tw_i = -1, -2$. That is, using the variable names in Fig. 1,* `history` *will include tuples of the following type: $(0, M_i, Z_i)$, $(+1, M_i, X_i)$, $(-1, M_i, X_i)$, $(+2, Y_i, Z_i)$ and $(-2, Y_i, Z_i)$. We say that $\mathcal{D}$ is a simplified adversary if the following two conditions hold:*

1. `history` *does not contain entries $(\pm 1, M_i, X_i)$ and $(2, Y_j, Z_j)$ with $i < j$ such that $Y_j = H_K(X_i)$.*
2. `history` *does not contain entries $(\pm 1, M_i, X_i)$, $(0, M_j, Z_j)$ and $(-2, Y_k, Z_k)$ with $i, j < k$ such that $M_j = M_i$ and $Z_k = Z_j$.*

where $(\pm 1, M_i, X_i)$ means that either $(+1, M_i, X_i)$ or $(-1, M_i, X_i)$ has been asked by $\mathcal{D}$ (note that $\mathcal{D}$ only asks one of these because pointless queries are disallowed). □

The first condition above means that $\mathcal{D}$ will not make a query $\mathcal{O}_2(Y)$ such that $Y = H_K(X)$ for an X which either was the response from a previously made query $\mathcal{O}_1(M)$ or was used in a previous query $\mathcal{O}_1^{-1}(X)$. The second condition means that $\mathcal{D}$ will not make a query $\mathcal{O}_2^{-1}(Z)$ such that Z was previously received as the answer for a query $\mathcal{O}_0(M)$ and M was either the response from a previously made query $\mathcal{O}_1^{-1}(X)$ or was used in a previous query $\mathcal{O}_1(M)$. Now refer to Fig. 2 where the complete simulated setting (Game I_0) is run with an arbitrary adversary A and the simplified simulated setting (Game I_1) is run with a simplified adversary $\mathcal{D}$. Comparing these two games, it can be seen that line 040 and line 050 of Game I_0 (which are responsible for handling the cases in which A may ask queries not conforming the two conditions in Definition 1) are omitted to get Game I_1. Now, as Game I_1 is only run with a simplified adversary $\mathcal{D}$ that must respect both of the two conditions in Definition 1, informally speaking, it is expected that $\mathbf{Adv}(\mathcal{D}^{G_0}, \mathcal{D}^{I_1}) = \mathbf{Adv}(A^{G_0}, A^{I_0})$. Lemma 1 provides a formal proof for this intuition. We note that our definition of a simplified adversary can be seen as an *extension* of the definition of a "construction-respecting" adversary from [27].

Lemma 1. *Let A be any pro adversary against our construction F that runs in time at most t and makes at most $(q_0, q_1, q_{-1}, q_2, q_{-2})$ queries to its five oracles $(\mathcal{O}_0, \mathcal{O}_1, \mathcal{O}_1^{-1}, \mathcal{O}_2, \mathcal{O}_2^{-1})$, respectively. Then we can construct a simplified adversary $\mathcal{D}$ such that*

$$\mathbf{Adv}(A^{G_0}, A^{I_0}) = \mathbf{Adv}(\mathcal{D}^{G_0}, \mathcal{D}^{I_1})$$

where $\mathcal{D}$ runs in time $t' \leq t + cq(\mathrm{Time}_H + log(q))$, for a small constant c, and makes at most $(q_0', q_1, q_{-1}, q_2, q_{-2})$ queries to its five oracles, where $q_0' = q_0 + q_1 + q_{-1}$. □

In the following lemmas, $\mathcal{D}$ is a simplified adversary.

Lemma 2. $\mathbf{Adv}(\mathcal{D}^{I_1}, \mathcal{D}^{I_3}) \leq \mathbf{Adv}_H^{CR}(B) + q\mathbf{Adv}_H^{OW}(C) + \frac{q}{2^m}$*, where the resources for adversaries B and C are as described in Theorem 1.* □

Lemma 3. $\mathbf{Adv}(\mathcal{D}^{G_0}, \mathcal{D}^{G_5}) \leq 1.5q^2\left(\frac{1}{2^m} + \frac{1}{2^n}\right) + q^2\Delta + \epsilon.$ □

Lemma 4. $\mathbf{Adv}(\mathcal{D}^{G_5}, \mathcal{D}^{I_3}) \leq q^2\left(\frac{2.5}{2^m} + \frac{2}{2^n} + \frac{1.5}{2^{n\mu(H)}} + \Delta\right).$ □

In the following subsections we prove Lemma 1 and Lemma 2. Proofs of Lemma 3 and Lemma 4 together with their related games are provided in Appendix.

Putting Pieces Together. Now we are ready to conclude the proof of Theorem 1. Combining lemmas 1–4 we have

$$\begin{aligned}\mathbf{Adv}_{F,\mathcal{S}}^{\mathrm{pro}}(A) = \mathbf{Adv}(A^{G_0}, A^{I_0}) &= \mathbf{Adv}(\mathcal{D}^{G_0}, \mathcal{D}^{I_1}) \\ &\leq \mathbf{Adv}(\mathcal{D}^{G_0}, \mathcal{D}^{G_5}) + \mathbf{Adv}(\mathcal{D}^{G_5}, \mathcal{D}^{I_3}) + \mathbf{Adv}(\mathcal{D}^{I_3}, \mathcal{D}^{I_1}) \\ &\leq \mathbf{Adv}_H^{CR}(B) + q\mathbf{Adv}_H^{OW}(C) + \epsilon + \frac{q}{2^m} + q^2\left(\frac{4}{2^m} + \frac{3.5}{2^n} + \frac{1.5}{2^{n\mu(H)}} + 2\Delta\right).\end{aligned}$$

4.1 Proof of Lemma 1

Given an arbitrary adversary A, we construct a simplified adversary $\mathcal{D}$ that runs A and includes the checks done by $\mathcal{S}$ in lines 040 and 050 of Game I_0. (Note that these checks are done by $\mathcal{S}$ to fool any adversary that might try to distinguish Game G_0 and Game I_0 by asking queries that disrespect the conditions of Definition 1). Adversary $\mathcal{D}$, given oracles $\mathcal{O}_0', \mathcal{O}_1', \mathcal{O}_1'^{-1}, \mathcal{O}_2', \mathcal{O}_2'^{-1}$ and the key K, runs $A(K)$ by answering A's oracle queries as shown in Fig. 3. If A makes $(q_0, q_1, q_{-1}, q_2, q_{-2})$ queries to its five oracles (let $q = q_0 + q_1 + q_{-1} + q_2 + q_{-2}$) then $\mathcal{D}$ makes $(q_0', q_1, q_{-1}, q_2, q_{-2})$ queries to its five oracles where $q_0' = q_0 + q_1 + q_{-1}$. Adversary $\mathcal{D}$ runs in time $t + cq(\mathrm{Time}_H + \log(q))$ where c is a small constant and $c\log(q)$ accounts for the maximum time required for searching an element in the memory $\mathcal{C}$ (note that $|\mathcal{C}| \leq q$).

Now it remains to show that

$$\Pr\left[\mathcal{D}^{G_0} \Rightarrow 1\right] = \Pr\left[A^{G_0} \Rightarrow 1\right], \text{and} \tag{1}$$

$$\Pr\left[\mathcal{D}^{I_1} \Rightarrow 1\right] = \Pr\left[A^{I_0} \Rightarrow 1\right]. \tag{2}$$

By construction of $\mathcal{D}$ (see Fig. 3) we have that A's query/response transcripts will be identical when it interacts directly with a game or it is run within $\mathcal{D}$ *unless* A asks queries that disrespect (contradict) one of the conditions required from a simplified adversary in Definition 1. (These types of queries are handled at lines 30 and 40 of $\mathcal{D}$ in Fig. 3.) Now we need to justify that even in the case of such disrespecting queries, A views identically distributed responses whether it is run within $\mathcal{D}$ or directly with the games G_0 and I_0. (The proof is essentially an extension of a similar argument in [27] for construction-respecting adversaries.) Assume that A asks queries that contradict the conditions in Definition 1. That is, we have:

Case 1. $(\pm 1, M, X) \in \mathtt{history}_A$ and A is making an $\mathcal{O}_2(H_K(X))$ query (i.e., A disrespects the first condition in Definition 1), or

Case 2. $(\pm 1, M, X) \in \mathtt{history}_A$ and $(0, M, Z) \in \mathtt{history}_A$, and A is making an $\mathcal{O}_2^{-1}(Z)$ query (i.e., A disrespects the second condition in Definition 1).

```
Adversary $\mathcal{D}(K)$
Run $A(K)$, answering its queries as follows:

on query $\mathcal{O}_0(M)$:
00  return $Z \leftarrow \mathcal{O}'_0(M)$

on query $\mathcal{O}_1(M)$:
10  $X \leftarrow \mathcal{O}'_1(M)$
11  $Y \leftarrow H_K(X)$
12  $Z \leftarrow \mathcal{O}'_0(M)$
13  $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{(M, X, Y, Z)\}$
14  return $X$

on query $\mathcal{O}_1^{-1}(X)$:
20  $M \leftarrow \mathcal{O}'^{-1}_1(X)$
21  $Z \leftarrow \mathcal{O}'_0(M)$
22  $Y \leftarrow H_K(X)$
23  $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{(M, X, Y, Z)\}$
24  return $M$

on query $\mathcal{O}_2(Y)$:
30  if $\exists\ (M, X, Y, Z) \in \mathcal{C}'$ then return $Z$
31  return $Z \leftarrow \mathcal{O}'_2(Y)$

on query $\mathcal{O}_2^{-1}(Z)$:
40  if $\exists\ (M, X, Y, Z) \in \mathcal{C}'$ then return $Y$
41  return $Y \leftarrow \mathcal{O}'^{-1}_2(Z)$

when $A$ halts with output bit $b$, output $b$.
```

Fig. 3. Constructing a simplified adversary $\mathcal{D}$ from an arbitrary adversary A.

To justify (1), first we consider Case 1 above. If A is run directly with Game G_0 then we have $X = \pi_1[M]$ and A receives $Z \leftarrow \pi_2(H_K(\pi_1[M]))$ as response for its $\mathcal{O}_2(H_K(X))$ query. If A is run within $\mathcal{D}$, which in turn has access to the oracles in Game G_0, then the condition at line 30 of Fig. 3 will be true and A receives a value Z from $\mathcal{C}'$ which is already associated to M (either at line 12 or line 21 in Fig. 3) as $Z \leftarrow \mathcal{O}'_0(M)$. Now, note that in Game G_0 the response for query $\mathcal{O}'_0(M)$ is also evaluated as $Z \leftarrow \pi_2(H_K(\pi_1[M]))$. So, the response Z will have identical distributions in the experiments A^{G_0} and $\mathcal{D}^{G_0}$ in this case.

Similarly, in Case 2, if A is run directly with Game G_0 we have $X = \pi_1[M]$ (hence, $M = \pi_1^{-1}[X]$), $Y = H_K(X)$, $Z = \pi_2[Y]$ (hence, $Y = \pi_2^{-1}[Z]$); therefore, A receives $Y \leftarrow \pi_2^{-1}[Z]$ as response for its $\mathcal{O}_2^{-1}[Z]$ query. On the other hand, if A is run within $\mathcal{D}$ then the condition at line 40 of Fig. 3 will be true and A receives a value Y from $\mathcal{C}'$ which is already associated to M, X, Z either in lines 10-13 (corresponding to $(+1, M, X) \in \texttt{history}_A$) or lines 20-23 (corresponding to $(-1, M, X) \in \texttt{history}_A$) in Fig. 3. Now, referring to the description of $\mathcal{D}$ in Fig. 3 and remembering that $\mathcal{D}$ is run with Game G_0, we have $Y = H_K(X)$ (line 11 or line 22), $Z \leftarrow \mathcal{O}'_0(M)$ (line 12 or line 21), and either $X \leftarrow \mathcal{O}'_1(M)$ (line 10) or $M \leftarrow \mathcal{O}'^{-1}_1(X)$ (line 20). That is, we have $X = \pi_1[M]$ (or equivalently, $M = \pi_1^{-1}[X]$) and $Z = \pi_2\left(H_K(\pi_1[M])\right) = \pi_2[Y]$, hence, $Y = \pi_2^{-1}[Z]$. Therefore, the response Y to the query $\mathcal{O}_2^{-1}(Z)$ will also have identical distributions in the experiments A^{G_0} and $\mathcal{D}^{G_0}$ in Case 2. So, we have $\Pr\left[\mathcal{D}^{G_0} \Rightarrow 1\right] = \Pr\left[A^{G_0} \Rightarrow 1\right]$.

To justify (2), we note that the simplified adversary D never makes queries of the types in Case 1 or Case 2 above; hence, we have $\Pr\left[\mathcal{D}^{I_0} \Rightarrow 1\right] = \Pr\left[\mathcal{D}^{I_1} \Rightarrow 1\right]$ (note that I_1 is the same as I_0 except that we have omitted the checks necessary to detect and handle queries causing Case 1 (at line 040 of Game I_0) and Case 2 (at line 040 of Game I_0). It remains to show that $\Pr\left[\mathcal{D}^{I_0} \Rightarrow 1\right] = \left[A^{I_0} \Rightarrow 1\right]$. The justification for this (by a straightforward case analysis) is very similar to the one we just used to show $\Pr\left[\mathcal{D}^{G_0} \Rightarrow 1\right] = \Pr\left[A^{G_0} \Rightarrow 1\right]$ and omitted here.

4.2 Proof of Lemma 2

Fig. 4 shows the sequence of games $I_1 \to I_2 \to I_3$ that we use to prove Lemma 2. Game I_1 includes the boxed statements (at lines 120, 130, and 132) while Game I_2 does not. We remind that the games are run with a simplified adversary and pointless queries are disallowed.

```
$\boxed{\text{Game } I_1}$
Game $I_2$

Procedure Initialize
$\mathcal{R} \xleftarrow{\$} \mathrm{Func}(m, n)$
return $K \xleftarrow{\$} \mathcal{K}$

Procedure $\mathcal{O}_0(M)$
return $Z \leftarrow \mathcal{R}[M]$

Procedure $\mathcal{O}_1(M)$
if $M \in \mathcal{P}_e$ then $bad \leftarrow \texttt{true}$, $\boxed{\textbf{return } \bot}$
$Z \leftarrow \mathcal{R}[M]$
if $\exists\, (\bot, X, Y, Z) \in \mathcal{C}$ then
    $\mathcal{C} \leftarrow (\mathcal{C} \setminus \{(\bot, X, Y, Z)\}) \cup \{(M, X, Y, Z)\}$
    $\mathcal{P}_d \leftarrow \mathcal{P}_d \setminus \{X\}$, return $X$
$X \xleftarrow{\$} \{0,1\}^m$
$Y \leftarrow H_K(X)$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(M, X, Y, Z)\}$
return $X$

Procedure $\mathcal{O}_1^{-1}(X)$
if $X \in \mathcal{P}_d$ then $bad \leftarrow \texttt{true}$, $\boxed{\textbf{return } \bot}$
$Y \leftarrow H_K(X)$
if $\exists\, (M', X', Y, Z') \in \mathcal{C} \wedge X \neq X'$ then $bad \leftarrow \texttt{true}$, $\boxed{\textbf{return } \bot}$
if $\exists\, (M, \bot, Y, Z) \in \mathcal{C}$ then
    $\mathcal{C} \leftarrow (\mathcal{C} \setminus \{(M, \bot, Y, Z)\}) \cup \{(M, X, Y, Z)\}$
    $\mathcal{P}_e \leftarrow \mathcal{P}_e \setminus \{M\}$, return $M$
$M \xleftarrow{\$} \{0,1\}^m$
$Z \leftarrow \mathcal{R}[M]$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(M, X, Y, Z)\}$
return $M$

Procedure $\mathcal{O}_2(Y)$
$M \xleftarrow{\$} \{0,1\}^m$
$Z \leftarrow \mathcal{R}[M]$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(M, \bot, Y, Z)\}$
$\mathcal{P}_e \leftarrow \mathcal{P}_e \cup \{M\}$
return $Z$

Procedure $\mathcal{O}_2^{-1}(Z)$
$X \xleftarrow{\$} \{0,1\}^m$
$Y \leftarrow H_K(X)$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(\bot, X, Y, Z)\}$
$\mathcal{P}_d \leftarrow \mathcal{P}_d \cup \{X\}$
return $Y$
```

```
Game $I_3$

Procedure Initialize
$\mathcal{R} \xleftarrow{\$} \mathrm{Func}(m, n)$
return $K \xleftarrow{\$} \mathcal{K}$

Procedure $\mathcal{O}_0(M)$
return $Z \leftarrow \mathcal{R}[M]$

Procedure $\mathcal{O}_1(M)$
$Z \leftarrow \mathcal{R}[M]$
if $\exists\, (\bot, X, Y, Z) \in \mathcal{C}$ then return $X$
return $X \xleftarrow{\$} \{0,1\}^m$

Procedure $\mathcal{O}_1^{-1}(X)$
$Y \leftarrow H_K(X)$
if $\exists\, (M, \bot, Y, Z) \in \mathcal{C}$ then return $M$
return $M \xleftarrow{\$} \{0,1\}^m$

Procedure $\mathcal{O}_2(Y)$
$M \xleftarrow{\$} \{0,1\}^m$
$Z \leftarrow \mathcal{R}[M]$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(M, \bot, Y, Z)\}$
return $Z$

Procedure $\mathcal{O}_2^{-1}(Z)$
$X \xleftarrow{\$} \{0,1\}^m$
$Y \leftarrow H_K(X)$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{(\bot, X, Y, Z)\}$
return $Y$
```

Fig. 4. Sequence of games used for proving Lemma 2. Game I_1 includes the boxed (**return** $\bot$) statements while Game I_2 does not.

$\boldsymbol{I_1 \rightarrow I_2}$**.** Games I_1 and I_2 are *identical-until-bad* and so from the fundamental lemma of game-playing [4] we have $\mathbf{Adv}(\mathcal{D}^{I_1}, \mathcal{D}^{I_2}) = \Pr\left[\mathcal{D}^{I_1} \text{ sets } bad\right]$. Now, we bound $\Pr\left[\mathcal{D}^{I_1} \text{ sets } bad\right]$ using the union bound and case analysis of the events in which *bad* might set to true in Game I_1.

Line 120. The flag *bad* is set at this line if $\mathcal{D}$ makes a query $\mathcal{O}_1(M)$ such that $M \in \mathcal{P}_e$. Now, note that the set of poisoned queries, $\mathcal{P}_e$, is generated as a result of queries to $\mathcal{O}_2$ and consists of randomly chosen values for M (at line 140 of Game I_1) about which adversary $\mathcal{D}$ only gets corresponding random values $Z \leftarrow \mathcal{R}[M]$ (see lines 141 and 144 of Game I_1), i.e. output values from a random function $\mathcal{R}$. If adversary can make the simulator reveal a poisoned value M at line 135 (which is possible by crafting an appropriate sequence of queries [1]) then M is deleted from $\mathcal{P}_e$ and will no longer be a poisoned value. Therefore, setting *bad* to true at line 120 in Game I_1 requires that adversary $\mathcal{D}$ guesses a random poisoned value $M \in \mathcal{P}_e$ which has never been revealed, by only having the corresponding random values $Z \leftarrow \mathcal{R}[M]$ for such unknown random values for M. As $\mathcal{R}$ is a random function and $|\mathcal{P}_e| \leq q_2$ we have

$$\Pr[\mathcal{D}^{I_1} \text{sets } bad \text{ at line } 120] \leq q_2 2^{-m}. \tag{3}$$

Line 130. The flag *bad* is set at this line if adversary $\mathcal{D}$ makes a query $\mathcal{O}_1^{-1}(X)$ such that $X \in \mathcal{P}_d$. The set of poisoned values for X, i.e. $\mathcal{P}_d$, is generated as a result of queries to $\mathcal{O}_2^{-1}$ and consists of randomly chosen values for X (at line 150 of Game I_1) about which $\mathcal{D}$ only gets the corresponding hash values $Y \leftarrow H_K(X)$ (see lines 150-154 of Game I_1). That is, each query to oracle $\mathcal{O}_2^{-1}$ provides the adversary with an image value $Y \leftarrow H_K(X)$ where the corresponding input value X is chosen at random and not revealed to the adversary. Adversary can make the simulator reveal some of these input values X at line 124 of Game I_1 (by crafting an appropriate sequence of queries [2]) in which case such values for X are deleted from $\mathcal{P}_d$ (hence, no longer will be relevant for the conditional statement at line 130). Therefore, to set *bad* at line 130 of Game I_1, adversary $\mathcal{D}$ must find one of the input values X (which is not revealed yet) for a given image value Y (obtained from a previous query to $\mathcal{O}_2^{-1}$). Therefore, the probability that adversary $\mathcal{D}$ can make *bad* be true at line 130 is bounded by the probability that $\mathcal{D}$ can win in the following experiment against the hash function H_K: adversary adaptively receives several images $\{Y_1, Y_2, \cdots, Y_Q\}$ under $H_K(.)$ for random inputs $\{X_1, X_2, \cdots, X_Q\}$; adversary gets to learn some of these inputs (but not all of them), and finally adversary must find one of the remaining (unrevealed) inputs, i.e., *invert* a remaining image value Y_i. (Note that inverting an image value Y implies finding a preimage for Y but the converse does not hold necessarily, as there may be several preimages for a given image value.) This experiment captures a notion that was called “some-point-one-way” function (spowf) by Ristenpart and Shrimpton in [26, 27]. A straightforward hybrid argument similar to one shown in [27] can be used to reduce a spowf adversary $\mathcal{D}$ to an OW adversary C such that $\mathbf{Adv}_H^{OW}(C) \geq \frac{1}{Q}.\mathbf{Adv}_H^{spowf}(\mathcal{D})$. Now, noticing that $|\mathcal{P}_d| \leq q_{-2}$ (equality holds if none of the poisoned values X are revealed at line 124 of Game I_1), we have

$$\Pr[\mathcal{D}^{I_1} \text{sets } bad \text{ at line } 130] \leq \mathbf{Adv}_H^{spowf}(\mathcal{D}) \leq q_{-2}\mathbf{Adv}_H^{OW}(C). \tag{4}$$

Line 132. The flag *bad* is set at this line if $\mathcal{D}$ makes a query $\mathcal{O}_1^{-1}(X)$ such that, under the hash function $H_K(.)$, X collides with an X' that was already stored by the simulator in a complete tuple in $\mathcal{C}$ and $X' \neq X$. Let B be an adversary that runs $\mathcal{D}$ and answers $\mathcal{D}$'s queries as described in Game I_1. Clearly, if $\mathcal{D}$ can set *bad* to true at line 132 then B will output the colliding pair (X, X') and wins the CR game against H_K. So, we have

$$\Pr[\mathcal{D}^{I_1} \text{sets } bad \text{ at line } 132] = \mathbf{Adv}_H^{CR}(B). \tag{5}$$

[1] An adversary that has an X and its hash value $Y = H_K(X)$ can make the simulator reveal a poisoned value M at line 135 by asking $\mathcal{O}_2(Y)$ followed by $\mathcal{O}_1^{-1}(X)$.

[2] An adversary, having an arbitrary M, can make the simulator reveal a poisoned value X at line 124 in Game I_1 by asking the following sequence of queries: $Z \leftarrow \mathcal{O}_0(M)$; $Y \leftarrow \mathcal{O}_2^{-1}(Z)$; and $X \leftarrow \mathcal{O}_1(M)$.

From 3, 4, and 5 we have

$$\mathbf{Adv}(\mathcal{D}^{I_1}, \mathcal{D}^{I_2}) \leq \mathbf{Adv}_H^{CR}(B) + q_{-2}\mathbf{Adv}_H^{OW}(C) + \frac{q_2}{2^m}. \tag{6}$$

$I_2 \longrightarrow I_3$. To move from I_2 to I_3, we omit parts of the code of I_2 which were responsible for handling the flag *bad* and the poisoned queries. This change is conservative as these operations (that were used for capturing the difference between games I_1 and I_2) are now redundant in Game I_2 and do not affect any other variables. So, we have

$$\mathbf{Adv}(\mathcal{D}^{I_2}, \mathcal{D}^{I_3}) = 0. \tag{7}$$

Now, from (6) and (7), and remembering that $q = q_0 + q_1 + q_{-1} + q_2 + q_{-2}$, we have

$$\mathbf{Adv}(\mathcal{D}^{I_1}, \mathcal{D}^{I_3}) \leq \mathbf{Adv}(\mathcal{D}^{I_1}, \mathcal{D}^{I_2}) + \mathbf{Adv}(\mathcal{D}^{I_2}, \mathcal{D}^{I_3}) \leq \mathbf{Adv}_H^{CR}(B) + q\mathbf{Adv}_H^{OW}(C) + \frac{q}{2^m}. \tag{8}$$

This completes the proof of Lemma 2. □

References

[1] Andreeva, E., Neven, G., Preneel, B., Shrimpton, T.: Seven-Property-Preserving Iterated Hashing: ROX. In: Kaoru Kurosawa (ed.): ASIACRYPT 2007. LNCS, vol. 4833, pp. 130–146. Springer (2007)

[2] Biham, E., Dunkelman, O.: A framework for iterative hash functions–HAIFA. Cryptology ePrint Report 2007/278, 2007.

[3] Bellare, M., Kohno, T.: Hash Function Balance and its Impact on Birthday Attacks. Cryptology ePrint Archive, Report 2003/065.

[4] Bellare, M., Rogaway, P.: Code-Based Game-Playing Proofs and the Security of Triple Encryption. Cryptology ePrint Archive, Report 2004/331.

[5] Bellare, M., Ristenpart, T.: Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In Lai, X., Chen, K. (eds.): ASIACRYPT 2006. LNCS, vol. 4284, pp. 299–314. Springer (2006)

[6] Bellare, M., Ristenpart, T.: Hash Functions in the Dedicated-Key Setting: Design Choices and MPP Transforms. In Arge, L., Cachin, C., Jurdzinski, T., Tarlecki, A. (eds.): ICALP 07. LNCS, vol. 4596, pp. 399–410. Springer (2007)

[7] Bellare, M., Ristov, T.: Hash Functions from Sigma Protocols and Improvements to VSH. In Pieprzyk, J. (ed.): ASIACRYPT 2008. LNCS, vol. 5350, pp. 125–142. Springer (2008)

[8] Bertoni, G., Daemen, J., Peeters, M., Van Assche G.: The Keccak sponge function family. Available at http://keccak.noekeon.org/.

[9] Black, J., Rogaway, P., Shrimpton, T., Stam, M.: An Analysis of the Blockcipher-Based Hash Functions from PGV. *J. Cryptology*, vol. 23, no. 4, pp. 519–545. Springer (2010)

[10] Charles, D.X., Lauter, K.E., Goren, E.Z.: Cryptographic Hash Functions from Expander Graphs. *J. Cryptology*, vol. 22, no. 1, pp. 93–113. Springer (2009)

[11] Contini, S., Lenstra, A.K., Steinfeld, R.: VSH, an Efficient and Provable Collision-Resistant Hash Function. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 165-182. Springer (2006)

[12] Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård Revisited: How to Construct a Hash Function. In Shoup, V. (ed.): CRYPTO 2005. LNCS, vol. 3621, pp. 430–448. Springer (2005)

[13] Coron, J-S., Dodis, Y., Mandal, A., Seurin, Y.: A Domain Extender for the Ideal Cipher. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 273-289. Springer (2010)

[14] Chang, D., Lee, S., Nandi, M., Yung, M.: Indifferentiable Security Analysis of Popular Hash Functions with Prefix-Free Padding. In Lai, X., Chen, K. (eds.): ASIACRYPT 2006. LNCS, vol. 4284, pp. 283–298. Springer (2006)

[15] Damgård, I.: Collision Free Hash Functions and Public Key Signature Schemes. In: Chaum, D., Price, W.L. (eds.) EUROCRYPT 1987. LNCS, vol. 304, pp. 203–216. Springer (1987)

[16] Fischlin, M., Lehmann, A., Pietrzak, K.: Robust Multi-property Combiners for Hash Functions Revisited. In Aceto, L., Damgård, I., and Goldberg, L.A., Halldórsson, M.M., and Ingólfsdóttir, A., Walukiewicz, I. (eds.): ICALP 2008, Part II. LNCS, vol. 5126, pp. 655–666. Springer (2008)

[17] Gauravaram, P., Knudsen, L., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.: Grøstl–a SHA-3 candidate. Available at http://www.groestl.info.

[18] Kaliski Jr., B. S.: One-Way Permutations on Elliptic Curves. *J. Cryptology*, vol. 3, no. 3, pp. 187-199. Springer (1991)

[19] Lehmann, A., Tessaro, S.: A Modular Design for Hash Functions: Towards Making the Mix-Compress-Mix Approach Practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 364-381. Springer (2009)

[20] Lyubashevsky, V., Micciancio, D., Peikert, C., Rosen, A.: SWIFFT: A Modest Proposal for FFT Hashing. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 54–72. Springer (2008)

[21] 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. LNCS, vol. 2951, pp. 21–39. Springer (2004).

[22] National Institute of Standards and Technology. Cryptographic Hash Algorithm Competition. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.

[23] Preneel, B.: Analysis and Design of Cryptographic Hash Functions. Doctoral dissertation, K. U. Leuven, 1993.

[24] Rogaway, P.: Formalizing Human Ignorance: Collision-Resistant Hashing without the Keys. In: Nguyen, P.Q. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 211–228. Springer (2006)

[25] Rogaway, P., Shrimpton, T.: Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In: Roy, B.K., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 371–388. Springer (2004)

[26] Ristenpart, T., Shrimpton, T.: How to Build a Hash Function from Any Collision-Resistant Function. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 147–163. Springer (2007)

[27] Ristenpart, T., Shrimpton, T.: How to Build a Hash Function from Any Collision-Resistant Function. Cryptology ePrint Archive, Report 2008/189.

[28] Rogaway, P., Steinberger, J.P.: Constructing Cryptographic Hash Functions from Fixed-Key Blockciphers. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 433450. Springer (2008)

[29] Ristenpart, T., Shacham, H., Shrimpton, T.: Careful with Composition: Limitations of the Indifferentiability Framework. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 487–506. Springer (2011)

[30] Shoup, V.: Sequences of Games: A Tool for Taming Complexity in Security Proofs. Cryptology ePrint report 2004/332.

[31] Saarinen, M-J. O.: Security of VSH in the Real World. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 95–103. Springer (2006)

[32] Wu, H.: The Hash Function JH. Available at http://www3.ntu.edu.sg/home/wuhj/research/jh/index.html.

A Appendix

A.1 Proof of Lemma 3

We use a sequence of games, $G_0 \to G_1 \to G_2 \to G_3 \to G_4 \to G_5$, to prove the lemma. G_0 is shown in Fig. 1, games $G_1 \to G_2 \to G_3 \to G_4$ are shown in Fig. 5, and G_5 is shown in Fig. 6. The transitions between these games are mainly based on basic and commonly used techniques in the game-playing framework [4]; hence, for lack of space, we only briefly overview them. The move from G_0 to G_1 we rewrite game G_0 by lazily growing π_1 and π_2. This is a conservative move and we have

$$\mathbf{Adv}(\mathcal{D}^{G_0}, \mathcal{D}^{G_1}) = 0. \tag{9}$$

Games G_1 and G_2 are identical-until-bad and from the fundamental lemma of game-playing technique we have $\mathbf{Adv}(\mathcal{D}^{G_1}, \mathcal{D}^{G_2}) = \Pr\left[\mathcal{D}^{G_1} \text{ sets } bad\right]$. Note that the flag *bad* is set in game G_1 whenever collisions happen during lazily growing π_1 and π_2. Using the standard birthday-bound, we have

$$\mathbf{Adv}(\mathcal{D}^{G_1}, \mathcal{D}^{G_2}) \leq \frac{0.5(q_0 + q_1 + q_{-1})^2}{2^m} + \frac{0.5(q_0 + q_2 + q_{-2})^2}{2^n} \leq \frac{0.5q^2}{2^m} + \frac{0.5q^2}{2^n}. \tag{10}$$

Games G_2 and G_3 are adversarially indistinguishable. To move from G_2 to G_3, we first omit the statements at lines 102, 122, 132, 142, and 152 of G_2 (note that G_2 does not include the boxed statements and hence omitting these lines will not affect adversary's view), then we rewrite the way that the tables π_1 and π_2 are handled in an equivalent way; namely, we first check whether a domain or range element is already defined in these tables and if so we return that element; otherwise, we go on by sampling a random point and defining the tables accordingly. (For the moment, setting *bad* at lines 330 and 340 of G_3 can be ignored as they will only be used later to move from G_3 to G_4.) Clearly, this code rewriting does not affect the distribution of the responses that an adversary gets in these two games, so we have

$$\mathbf{Adv}(\mathcal{D}^{G_2}, \mathcal{D}^{G_3}) = 0. \tag{11}$$

Games G_3 and G_4 are identical-until-bad, so we have $\mathbf{Adv}(\mathcal{D}^{G_3}, \mathcal{D}^{G_4}) = \Pr\left[\mathcal{D}^{G_3} \text{ sets } bad\right]$. We claim that $\Pr\left[\mathcal{D}^{G_3} \text{ sets } bad \text{ at line } 330\right] \leq \frac{q_0 q_{-1}}{2^m}$. Remembering that $\mathcal{D}$ is a *simplified adversary* and *pointless queries* are not allowed, we note that *bad* is set at line 330 if a new (non-redundant) query X made by the adversary to oracle $\mathcal{O}_1^{-1}$ *equals* to an already defined but *unrevealed* image element of π_1. Now, note that such already defined but unrevealed values for X (image elements for π_1) can only be generated as a result of queries to $\mathcal{O}_0$ (see lines 300-302); i.e., there can be at most q_0 such values and the probability to hit one of them in a query $\mathcal{O}_1^{-1}(X)$ (at line 330) is at most $\frac{q_0}{2^m}$. As adversary can make q_{-1} (non-redundant) queries to oracle $\mathcal{O}_1^{-1}$, using the union bound, the probability that adversary can set *bad* at line 330 is bounded to $\frac{q_0 q_{-1}}{2^m}$.

Now, it remains to bound $\Pr\left[\mathcal{D}^{G_3} \text{ sets } bad \text{ at line } 340\right]$; this is the probability that a new (non-redundant) query Y made by the adversary to oracle $\mathcal{O}_2$ collides with an already defined but unrevealed domain element of π_2. Such already defined but unrevealed values for Y are the outputs of the hash function H_K (see Fig. 1); i.e., the intermediate values about which a simplified adversary is not given any information. So, we can bound the probability that adversary can set *bad* at line 340 by the probability that adversary can win the following combinatorial experiment: a random key $K \xleftarrow{\$} \mathcal{K}$ is selected and given to the adversary; adversary gets to choose any q_2 points $Y_1, Y_2, \cdots, Y_{q_2}$ from $\{0,1\}^n$ and let $\mathcal{Y} = \{Y_1, Y_2, \cdots, Y_{q_2}\}$; random values $X_i \xleftarrow{\$} \{0,1\}^m$ are chosen, for $1 \leq i \leq q_0$, and let $Y_i' = H_K(X_i)$; adversary wins if $Y_i' = Y_j$ for some $i \in \{1, \cdots, q_0\}$ and $j \in \{1, \cdots, q_2\}$. That is, we have $\Pr\left[\mathcal{D}^{G_3} \text{ sets } bad \text{ at line } 340\right] \leq \Pr\left[H_K(X_i) = Y_j \text{ for some i, j}\right]$. The latter probability (of the success in the combinatorial experiment) was calculated by Ristenpart and Shrimpton in [27] (and sounds to be the reason behind the definition of the Δ-regularity notion in [26, 27]); we use the known bound for this probability from (page 16 of) [27] by replacing appropriate parameters here and omit the calculations (more complete proofs are left to the full version of this paper). Namely, we have $\Pr\left[H_K(X_i) = Y_j \text{ for some i, j}\right] \leq \frac{q_0 q_2}{2^n} + q_0 q_2 \Delta$. Therefore, we have

$$\Pr\left[\mathcal{D}^{G_3} \text{ sets } bad\right] \leq \frac{q_0 q_{-1}}{2^m} + \frac{q_0 q_2}{2^n} + q_0 q_2 \Delta \leq \frac{q^2}{2^m} + \frac{q^2}{2^n} + q^2 \Delta. \tag{12}$$

To move from G_4 to G_5, we first omit the statements setting *bad* in Game G_4 (as these do not affect the responses in G_4). Then instead of selecting a random value Y directly from $\{0,1\}^n$ (at line 351 in G_4) we choose a random value X from the domain of the hash function H_K and set $Y = H_K(X)$ (at line 551 in G_5). As we assume that H is an ϵ-almost output regular hash function, we have $\mathbf{Adv}(\mathcal{D}^{G_4}, \mathcal{D}^{G_5}) \leq \epsilon$. Combing this and (9), (10), (11) and (12), we can conclude the proof of Lemma 3 as

$$\mathbf{Adv}(\mathcal{D}^{G_0}, \mathcal{D}^{G_5}) \leq 1.5q^2 \left(\frac{1}{2^m} + \frac{1}{2^n}\right) + q^2 \Delta + \epsilon.$$

A.2 Proof of Lemma 4

The proof is divided into three lemmas. In the following, we provide the lemmas together with the sequences of games, in Fig. 6 and Fig. 7, used for proving them. Descriptions of the proofs using the shown sequences of games and standard techniques to bound the movements between the games are straightforward and omitted here.

Lemma 5. *Let $\mathcal{D}$ be any simplified adversary that runs in time at most t and makes at most q queries to all of its oracles. We can construct an adversary $\mathcal{D}^*$ as shown in Fig. 7 such that*

$$\mathbf{Adv}(\mathcal{D}^{G_5}, \mathcal{D}^{I_3}) \leq \mathbf{Adv}(\mathcal{D}^{*G_6}, \mathcal{D}^{*I_4}) + \frac{q_0 q_{-2}}{2^n} + \frac{0.5(q_0 + q_{-2})^2}{2^{n\mu(H)}}$$

where $\mathcal{D}^*$ runs in time $t^* \leq t + cq(\mathrm{Time}_H + log(q))$, for a small constant c, and makes at most q queries to its own five oracles $(\mathcal{O}_0^*, \mathcal{O}_1^*, \mathcal{O}_1^{-1*}, \mathcal{O}_2^*, \mathcal{O}_2^{-1*})$. □

Proof (Sketch). Referring to the construction of $\mathcal{D}^*$ in Fig. 7, and the games I_3 (in Fig. 4), I_4 (in Fig. 6), G_5 and G_6 (in Fig. 6), we have

$$\Pr\left[\mathcal{D}^{*I_4} \Rightarrow 1\right] = \Pr\left[\mathcal{D}^{I_3} \Rightarrow 1 \wedge \overline{\textbf{Abort}}\right] \geq \Pr\left[\mathcal{D}^{I_3} \Rightarrow 1\right] - \frac{q_0 q_{-2}}{2^n} \tag{13}$$

$$\Pr\left[\mathcal{D}^{*G_6} \Rightarrow 1\right] = \Pr\left[\mathcal{D}^{G_5} \Rightarrow 1 \wedge \overline{\textbf{Abort}}\right] \geq \Pr\left[\mathcal{D}^{G_5} \Rightarrow 1\right] - \frac{0.5(q_0 + q_{-2})^2}{2^{n\mu(H)}}. \tag{14}$$

Using (13) and (14) we have

$$\begin{aligned}\textbf{Adv}(\mathcal{D}^{G_5}, \mathcal{D}^{I_3}) &= \left|\Pr\left[\mathcal{D}^{G_5} \Rightarrow 1\right] - \Pr\left[\mathcal{D}^{I_3} \Rightarrow 1\right]\right| \\ &\leq \left|\Pr\left[\mathcal{D}^{*G_6} \Rightarrow 1\right] - \Pr\left[\mathcal{D}^{*I_4} \Rightarrow 1\right]\right| + \frac{q_0 q_{-2}}{2^n} + \frac{0.5(q_0 + q_{-2})^2}{2^{n\mu(H)}} \\ &= \textbf{Adv}(\mathcal{D}^{*G_6}, \mathcal{D}^{*I_4}) + \frac{q_0 q_{-2}}{2^n} + \frac{0.5(q_0 + q_{-2})^2}{2^{n\mu(H)}}.\end{aligned}$$

□

Lemma 6. $\textbf{Adv}(\mathcal{D}^{*G_6}, \mathcal{D}^{*I_4}) \leq \textbf{Adv}(\mathcal{D}^{*G_7}, \mathcal{D}^{*I_8}) + \frac{2q_0 q_2}{2^m} + \frac{0.5q_2^2}{2^m}$. □

Sequences of games $I_4 \rightarrow I_5 \rightarrow I_6 \rightarrow I_7 \rightarrow I_8$ and $G_6 \rightarrow G_7$, and description of D^* are shown in Fig. 6 and Fig. 7.

Lemma 7. $\textbf{Adv}(\mathcal{D}^{*G_7}, \mathcal{D}^{*I_8}) \leq \frac{0.5q_0^2}{2^{n\mu H}} + \frac{0.5(q_0+q_{-2})^2}{2^{n\mu(H)}} + \frac{q_0 q_2}{2^n} + q_0 q_2 \varDelta$. □

Combining the bounds in lemmas 5–7 and noticing that $q_i \leq q$ for $i \in \{0, 1, -1, 2, -2\}$ (where q denotes the total number of queries by the adversary), we have

$$\textbf{Adv}(\mathcal{D}^{G_5}, \mathcal{D}^{I_3}) \leq q^2 \left(\frac{2.5}{2^m} + \frac{2}{2^n} + \frac{1.5}{2^{n\mu(H)}} + \varDelta\right). \tag{15}$$

This completes the proof of Lemma 4. □

```
$\boxed{\text{Game } G_1}$
Game $G_2$

Procedure Initialize
return $K \xleftarrow{\$} \mathcal{K}$

Procedure $\mathcal{O}_0(M)$
if $\pi_1[M] = \perp$ then
    $X \xleftarrow{\$} \{0,1\}^m$
    if $X \in \text{image}(\pi_1)$ then $bad \leftarrow$ true, $\boxed{X \xleftarrow{\$} \overline{\text{image}}(\pi_1)}$
    $\pi_1[M] \leftarrow X$
$X \leftarrow \pi_1[M]$
$Y \leftarrow H_K(X)$
if $\pi_2[Y] = \perp$ then
    $Z \xleftarrow{\$} \{0,1\}^n$
    if $Z \in \text{image}(\pi_2)$ then $bad \leftarrow$ true, $\boxed{Z \xleftarrow{\$} \overline{\text{image}}(\pi_2)}$
    $\pi_2[Y] \leftarrow Z$
$Z \leftarrow \pi_2[Y]$
return $Z$

Procedure $\mathcal{O}_1(M)$
if $\pi_1[M] = \perp$ then
    $X \xleftarrow{\$} \{0,1\}^m$
    if $X \in \text{image}(\pi_1)$ then $bad \leftarrow$ true, $\boxed{X \xleftarrow{\$} \overline{\text{image}}(\pi_1)}$
    $\pi_1[M] \leftarrow X$
$X \leftarrow \pi_1[M]$
return $X$

Procedure $\mathcal{O}_1^{-1}(X)$
if $\pi_1^{-1}[X] = \perp$ then
    $M \xleftarrow{\$} \{0,1\}^m$
    if $M \in \text{domain}(\pi_1)$ then $bad \leftarrow$ true, $\boxed{M \xleftarrow{\$} \overline{\text{domain}}(\pi_1)}$
    $\pi_1[M] \leftarrow X$
$M \leftarrow \pi_1^{-1}[X]$
return $M$

Procedure $\mathcal{O}_2(Y)$
if $\pi_2[Y] = \perp$ then
    $Z \xleftarrow{\$} \{0,1\}^n$
    if $Z \in \text{image}(\pi_2)$ then $bad \leftarrow$ true, $\boxed{Z \xleftarrow{\$} \overline{\text{image}}(\pi_2)}$
    $\pi_2[Y] \leftarrow Z$
$Z \leftarrow \pi_2[Y]$
return $Z$

Procedure $\mathcal{O}_2^{-1}(Z)$
if $\pi_2^{-1}[Z] = \perp$ then
    $Y \xleftarrow{\$} \{0,1\}^n$
    if $Y \in \text{domain}(\pi_2)$ then $bad \leftarrow$ true, $\boxed{Y \xleftarrow{\$} \overline{\text{domain}}(\pi_2)}$
    $\pi_2[Y] \leftarrow Z$
$Y \leftarrow \pi_2^{-1}[Z]$
return $Y$
```

```
$\boxed{\text{Game } G_3}$
Game $G_4$

Procedure Initialize
return $K \xleftarrow{\$} \mathcal{K}$

Procedure $\mathcal{O}_0(M)$
$X \xleftarrow{\$} \{0,1\}^m$
if $\pi_1[M] \neq \perp$ then $X \leftarrow \pi_1[M]$
$\pi_1[M] \leftarrow X$
$Y \leftarrow H_K(X)$
if $\pi_2[Y] \neq \perp$ then return $Z \leftarrow \pi_2[Y]$
$Z \xleftarrow{\$} \{0,1\}^n$
$\pi_2[Y] \leftarrow Z$
return $Z$

Procedure $\mathcal{O}_1(M)$
if $\pi_1[M] \neq \perp$ then return $X \leftarrow \pi_1[M]$
$X \xleftarrow{\$} \{0,1\}^m$
$\pi_1[M] \leftarrow X$
return $X$

Procedure $\mathcal{O}_1^{-1}(X)$
if $\pi_1^{-1}[X] \neq \perp$ then $bad \leftarrow$ true, $\boxed{\text{return } M \leftarrow \pi_1^{-1}[X]}$
$M \xleftarrow{\$} \{0,1\}^m$
$\pi_1[M] \leftarrow X$
return $M$

Procedure $\mathcal{O}_2(Y)$
if $\pi_2[Y] \neq \perp$ then $bad \leftarrow$ true, $\boxed{\text{return } Z \leftarrow \pi_2[Y]}$
$Z \xleftarrow{\$} \{0,1\}^n$
$\pi_2[Y] \leftarrow Z$
return $Z$

Procedure $\mathcal{O}_2^{-1}(Z)$
if $\pi_2^{-1}[Z] \neq \perp$ then return $Y \leftarrow \pi_2^{-1}[Z]$
$Y \xleftarrow{\$} \{0,1\}^n$
$\pi_2[Y] \leftarrow Z$
return $Y$
```

Fig. 5. Sequence of games used in the proof of Lemma 3. G_1 and G_3 include the boxed statements while G_2 and G_4 do not.

Game G_5

Procedure Initialize

```
000 return $K \xleftarrow{\$} \mathcal{K}$
```

Procedure $\mathcal{O}_0(M)$

```
500 $X \xleftarrow{\$} \{0,1\}^m$
501 if $\pi_1[M] \neq \perp$ then $X \leftarrow \pi_1[M]$
502 $\pi_1[M] \leftarrow X$
503 $Y \leftarrow H_K(X)$
504 if $\pi_2[Y] \neq \perp$ then return $Z \leftarrow \pi_2[Y]$
505 $Z \xleftarrow{\$} \{0,1\}^n$
506 $\pi_2[Y] \leftarrow Z$
507 return $Z$
```

Procedure $\mathcal{O}_1(M)$

```
520 if $\pi_1[M] \neq \perp$ then return $X \leftarrow \pi_1[M]$
521 $X \xleftarrow{\$} \{0,1\}^m$
522 $\pi_1[M] \leftarrow X$
523 return $X$
```

Procedure $\mathcal{O}_1^{-1}(X)$

```
530 $M \xleftarrow{\$} \{0,1\}^m$
531 $\pi_1[M] \leftarrow X$
532 return $M$
```

Procedure $\mathcal{O}_2(Y)$

```
540 $Z \xleftarrow{\$} \{0,1\}^n$
541 $\pi_2[Y] \leftarrow Z$
542 return $Z$
```

Procedure $\mathcal{O}_2^{-1}(Z)$

```
550 if $\pi_2^{-1}[Z] \neq \perp$ then return $Y \leftarrow \pi_2^{-1}[Z]$
551 $X \xleftarrow{\$} \{0,1\}^m$, $Y \leftarrow H_K(X)$
552 $\pi_2[Y] \leftarrow Z$
553 return $Y$
```

Game G_6

Procedure Initialize

```
000 return $K \xleftarrow{\$} \mathcal{K}$
```

Procedure $\mathcal{O}_0(M)$

```
600 $X \xleftarrow{\$} \{0,1\}^m$
601 if $\pi_1[M] \neq \perp$ then $X \leftarrow \pi_1[M]$
602 $\pi_1[M] \leftarrow X$
603 $Y \leftarrow H_K(X)$
604 if $\pi_2[Y] \neq \perp$ then return $Z \leftarrow \pi_2[Y]$
605 $Z \xleftarrow{\$} \{0,1\}^n$
606 $\pi_2[Y] \leftarrow Z$
607 return $Z$
```

Procedure $\mathcal{O}_1(M)$

```
320 $X \xleftarrow{\$} \{0,1\}^m$
321 $\pi_1[M] \leftarrow X$
322 return $X$
```

Procedure $\mathcal{O}_1^{-1}(X)$

```
630 $M \xleftarrow{\$} \{0,1\}^m$
631 $\pi_1[M] \leftarrow X$
632 return $M$
```

Procedure $\mathcal{O}_2(Y)$

```
640 $Z \xleftarrow{\$} \{0,1\}^n$
641 $\pi_2[Y] \leftarrow Z$
642 return $Z$
```

Procedure $\mathcal{O}_2^{-1}(Z)$

```
650 $X \xleftarrow{\$} \{0,1\}^m$, $Y \leftarrow H_K(X)$
651 $\pi_2[Y] \leftarrow Z$
652 return $Y$
```

$\boxed{\text{Game } I_4}$
Game I_5

Procedure Initialize

```
000 $\mathcal{R} \xleftarrow{\$} \mathrm{Func}(m,n)$
001 return $K \xleftarrow{\$} \mathcal{K}$
```

Procedure $\mathcal{O}_1(M)$

```
420 return $X \xleftarrow{\$} \{0,1\}^m$
```

Procedure $\mathcal{O}_2(Y)$

```
440 $M \xleftarrow{\$} \{0,1\}^m$
441 $Z \leftarrow \mathcal{R}[M]$
442 $\mathcal{C} \leftarrow \mathcal{C} \cup \{(M,\perp,Y,Z)\}$
443 return $Z$
```

Procedure $\mathcal{O}_0(M)$

```
410 return $Z \leftarrow \mathcal{R}[M]$
```

Procedure $\mathcal{O}_1^{-1}(X)$

```
430 $Y \leftarrow H_K(X)$
431 if $\exists\,(M,\perp,Y,Z) \in \mathcal{C}$ then $\boxed{\text{return } M}$
432 return $M \xleftarrow{\$} \{0,1\}^m$
```

Procedure $\mathcal{O}_2^{-1}(Z)$

```
450 $X \xleftarrow{\$} \{0,1\}^m$
451 $Y \leftarrow H_K(X)$
452 return $Y$
```

Fig. 6. Games used in the proof of Lemma 3 and Lemma 4. Game I_4 includes the boxed (return M) statement at line 431 while in I_5 it is omitted (i.e., replaced by an *empty* statement).

$\boxed{\text{Game } I_6}$
Game I_7

```
Procedure Initialize
  return $K \xleftarrow{\$} \mathcal{K}$

Procedure $\mathcal{O}_1(M)$
  return $X \xleftarrow{\$} \{0,1\}^m$

Procedure $\mathcal{O}_2(Y)$
  $M \xleftarrow{\$} \{0,1\}^m$
  $Z \xleftarrow{\$} \{0,1\}^n$
  if $\mathcal{R}[M] \neq \bot$ then $bad \leftarrow \texttt{true}$, $\boxed{Z \leftarrow \mathcal{R}[M]}$
  return $\mathcal{R}[M] \leftarrow Z$

Procedure $\mathcal{O}_0(M)$
  $Z \xleftarrow{\$} \{0,1\}^n$
  if $\mathcal{R}[M] \neq \bot$ then $bad \leftarrow \texttt{true}$, $\boxed{Z \leftarrow \mathcal{R}[M]}$
  return $\mathcal{R}[M] \leftarrow Z$

Procedure $\mathcal{O}_1^{-1}(X)$
  return $M \xleftarrow{\$} \{0,1\}^m$

Procedure $\mathcal{O}_2^{-1}(Z)$
  $X \xleftarrow{\$} \{0,1\}^m$
  $Y \leftarrow H_K(X)$
  return $Y$
```

$\boxed{\text{Game } G_7}$
Game I_8

```
Procedure Initialize
  return $K \xleftarrow{\$} \mathcal{K}$

Procedure $\mathcal{O}_0(M)$
  $X \xleftarrow{\$} \{0,1\}^m$
  if $\pi_1[M] \neq \bot$ then $X \leftarrow \pi_1[M]$
  $\pi_1[M] \leftarrow X$
  $Y \leftarrow H_K(X)$
  if $\pi_2[Y] \neq \bot$ then $bad \leftarrow \texttt{true}$, $\boxed{\text{return } Z \leftarrow \pi_2[Y]}$
  $Z \xleftarrow{\$} \{0,1\}^n$
  $\pi_2[Y] \leftarrow Z$
  return $Z$

Procedure $\mathcal{O}_1(M)$
  $X \xleftarrow{\$} \{0,1\}^m$
  $\pi_1[M] \leftarrow X$
  return $X$

Procedure $\mathcal{O}_1^{-1}(X)$
  $M \xleftarrow{\$} \{0,1\}^m$
  $\pi_1[M] \leftarrow X$
  return $M$

Procedure $\mathcal{O}_2(Y)$
  $Z \xleftarrow{\$} \{0,1\}^n$
  $\pi_2[Y] \leftarrow Z$
  return $Z$

Procedure $\mathcal{O}_2^{-1}(Z)$
  $X \xleftarrow{\$} \{0,1\}^m$, $Y \leftarrow H_K(X)$
  $\pi_2[Y] \leftarrow Z$
  return $Y$
```

Adversary $\mathcal{D}^*(K)$
Run $\mathcal{D}(K)$, answering its queries as follows:

```
on query $\mathcal{O}_0(M)$:
  $Z \leftarrow \mathcal{O}_0^*(M)$
  if $Z \in \mathcal{P}_z$ then Abort
  $\mathcal{S}_0 \leftarrow \mathcal{S}_0 \cup \{(M,Z)\}$
  return $Z$

on query $\mathcal{O}_1^{-1}(X)$:
  return $M \leftarrow \mathcal{O}_1^{-1*}(X)$

on query $\mathcal{O}_2(Y)$:
  return $Z \leftarrow \mathcal{O}_2^*(Y)$

on query $\mathcal{O}_1(M)$:
  if $\exists\,(M,Z) \in \mathcal{S}_0 \wedge \exists\,(Y,Z) \in \mathcal{S}_2$ then
      return $X \leftarrow \texttt{YtoX}[Y]$
  return $X \leftarrow \mathcal{O}_1^*(M)$

on query $\mathcal{O}_2^{-1}(Z)$:
  if $\exists\,(M,Z) \in \mathcal{S}_0$ then
      $X \xleftarrow{\$} \{0,1\}^m$ , $Y \leftarrow H_K(X)$
      $\texttt{YtoX}[Y] \leftarrow X$, $\mathcal{S}_2 \leftarrow \mathcal{S}_2 \cup \{(Y,Z)\}$
      return $Y$
  $\mathcal{P}_z \leftarrow \mathcal{P}_z \cup \{Z\}$
  return $Y \leftarrow \mathcal{O}_2^{-1*}(Z)$
```

when $\mathcal{D}$ halts with output bit b, output b.

Fig. 7. Games used in the proof of Lemma 4, and (at the bottom) building a construction-respecting simplified adversary $\mathcal{D}^*$ from any arbitrary simplified adversary $\mathcal{D}$ as used in Lemma 5.