A Unified Indifferentiability Proof for Permutation- or Block Cipher-Based Hash Functions

Anne Canteaut[2,3], Thomas Fuhr[1],
María Naya-Plasencia[4], Pascal Paillier[5], Jean-René Reinhard[1], Marion Videau[6]

[1] ANSSI
[2] INRIA Paris-Rocquencourt
[3] Technical University of Denmark
[4] University of Versailles-Saint Quentin en Yvelines
[5] CryptoExperts
[6] University of Lorraine, CNRS, INRIA Nancy-Grand Est / LORIA

Abstract. In the recent years, several hash constructions have been introduced that aim at achieving enhanced security margins by strengthening the Merkle-Damgård mode. However, their security analysis have been conducted independently and using a variety of proof methodologies. This paper unifies these results by proposing a unique indifferentiability proof that considers a broadened form of the general compression function introduced by Stam at FSE09. This general definition enables us to capture in a realistic model most of the features of the mode of operation (*e.g.*, message encoding, blank rounds, message insertion,...) within the pre-processing and post-processing functions. Furthermore, it relies on an inner primitive which can be instantiated either by an ideal block cipher, or by an ideal permutation. Then, most existing hash functions can be seen as the Chop-MD construction applied to some compression function which fits the broadened Stam model. Our result then gives the tightest known indifferentiability bounds for several general modes of operations, including Chop-MD, Haifa or sponges. Moreover, we show that it applies in a quite automatic way, by providing the security bounds for 7 out of the 14 second round SHA-3 candidates, which are in some cases improved over previously known ones.

Keywords: hash function, indifferentiability, SHA-3.

1 Introduction

The vast majority of hash function designs rely on the iteration of a compression function on a sequence of message blocks, based on the principle introduced by Merkle and Damgård [29, 22]. However, the publication since 2004 of new generic attacks on the Merkle-Damgård (MD) construction (multi-collisions, 2nd-preimage attacks for long messages, herding attacks...) has boosted the quest for improvements of the original MD mode that would reach a significantly enhanced level of resistance. These variants have opted for various strategies: doubling the internal state size [26], ensuring a prefix-free message encoding by adding a final stage or a block counter in the message [21], making use of a block counter whose treatment differs from the other input bits [12]... They have also been combined with the different ways to derive a compression function from a block cipher, including the well-known PGV modes [30]. Even much deeper modifications have been proposed: instead of an internal block cipher, the sponge construction [8] uses an internal unkeyed permutation.

All these works have led to a huge variety of modes of operation, as reflected by the design choices made by the 64 submissions to the SHA-3 competition. But, all these constructions obviously do not provide the same security level and the security of some of them remain somewhat obscure. In order to guarantee that their proposal is based on a sound construction, the designers should then provide some security arguments and prove that the way the internal block cipher or permutation is iterated does not introduce any weakness in the hash function. For this purpose, the best studied modes of operation have been proved to be indifferentiable from a random oracle: indifferentiability results for several variants of the Merkle-Damgård construction are given in [21] and then refined in [19, 18], the case of Haifa has been investigated in [10] while similar results for the sponge construction and its generalization can be found in [9, 6]. Also, some of the SHA-3 proposals include an ad-hoc indifferentiability proof dedicated to the underlying particular mode of operation, *e.g.*, [7, 16, 3, 11].

The dispersion of these results and the fact that these proofs are highly technical and hard to check make it difficult to evaluate the security achieved by some of these constructions: collecting all related results clearly requires an important effort [3, 1]. Also, the obtained indifferentiability bounds are relevant only with a clear description of the components of the hash function which have been idealized in the proof. This situation also makes it impossible to compare the respective advantages of the different modes of operation, and quantifying the security brought by some features, like adding a block counter to a fraction of the chaining variable, or inserting blank rounds, is still an open problem in some cases. Moreover, the lack of a synthetic and comprehensive view of these results restrains the possibilities to capture the influence of some non ideal properties of the internal primitives on the security of the mode of operation. This line of work, introduced in [17], and followed in [15, 2], has not yet been furthered satisfactorily, surely due to the already technical and complex proofs that had to be adapted to include these non-random properties. Then, it appears that any progress in evaluating the security achieved by all these constructions and their variants requires a global and synthetic approach. An important step in this direction was taken by Stam [32] who introduced a very general form for blockcipher-based compression functions, which encompasses the PGV compression functions. The collision- and preimage-resistance of PGV constructions had been studied by Black *et al.* in [13]. Stam then generalized this study to his more generic framework. He obtained [32, 14], under some assumptions on the pre-processing and post-processing parts of the compression function, general results on the resistance of blockcipher-based modes to collision and preimage attacks. A similar approach has been followed in [6] in the case of compression functions based on an unkeyed permutation. For this generalization of the sponge construction, named the parazoa family, an indifferentiability proof is given based similar assumptions.

Our contributions. In this paper, we consider a broadened version of Stam's compression function, where we allow the internal block cipher to have a zero-bit key, *i.e.*, this extension includes compression functions based on an internal block cipher and on an internal permutation. We show that this type of compression function combined with the Chop-MD construction can be used to represent most of the known hash functions in a realistic way, even when they include some particular features like counters or final rounds. For instance, we show how 9 out of 14 second-round candidates in the SHA-3 competition fit this broadened description. Moreover, it allows to clearly separate the primitive which is expected to have an ideal behaviour, from some simple pre- or post-computation, like the insertion of a counter, which may artificially cause some non-ideal behaviour, as noted in [20, 2] in the case of BLAKE. Then, we provide a general indifferentiability proof for this broadened mode of operation, leading to a security bound which depends on the choice of the padding, and of the pre-processing and post-processing parts of the compression function. In particular, the proof technique is the same for blockcipher-based and permutation-based compression functions.

This unified result points out that the difficult part in such an indifferentiability proof is to evaluate the resistance of the construction against the distinguishing attacks where the adversary detects some inconsistencies by first querying some hash values, and then calling the inner primitive. Length-extension attacks are typical examples of those techniques and illustrate the fact that the program simulating the inner primitive does not have any knowledge of the calls made to the hash function. Here, we show that this particular issue introduces some difficulties in the proofs, which are sometimes overlooked as noted in [16] and in [23, Page 13], in the case of wide-state constructions [9, 6] where the complexities of those attacks appear to be negligible. A similar flaw in the indifferentiability proofs of PGV modes with prefix-free padding [21, 19] has been identified in [27, Section 4]. Here, we introduce a new sequence of games which allows to handle this issue carefully.

Another consequence of our proof is that it leads to a bound which can then be applied in an automatic way to any hash function that fits the broadened model, if the blocksize of the (possibly keyed) permutation is higher than the size of the digest. It includes several cases which are examined for the first time, in particular because it does not require that the block size of the underlying ideal primitive be equal to the size of the chaining variable, which is an assumption for the parazoa family or for most previous results on blockcipher-based constructions. Then, for most of the second round SHA-3 candidates, we derive similar or even improved indifferentiability bounds compared to the results collected in [1, 3, 4].

2 Hash function based on the broadened Stam compression function

In [32], a general representation of blockcipher-based compression functions has been introduced by Stam. This generalization was used to classify and identify the common properties of the PGV hashing modes that could provide collision resistance. Under some specific properties of the underlying components, some bounds could be given, and the preimage resistance in some cases could be analyzed. Here, we generalize further this representation of a compression function by considering not only blockcipher-based compression functions but also compression functions based on permutations. Moreover, we need less restrictive assumptions on the underlying components of the compression function than those required in [32] and in [6]. For instance, we do not require any particular relationship between the block size of the (possibly keyed) permutation and the size of the chaining variable. A very important feature of a general representation for the compression function is that, together with an appropriate padding scheme, it can include all the features of most existing modes of operation (*e.g.*, a block counter, final rounds...). Then, most hash functions can be defined by such a general compression function, a padding rule and the plain (or chopped) Merkle-Damgård mode of operation.

2.1 Broadening Stam general compression function

The general representation of a blockcipher-based compression function introduced by Stam [32] is depicted on Figure 1. The input of the compression function consists of an s-bit chaining value V and an m-bit message block M. The corresponding output is a new s-bit chaining value W. The inputs of the compression function are first combined by a pre-processing function C^{PRE} for generating the k-bit key K and the n-bit input X of the block cipher F. The n-bit output Y of the block cipher, as well as M and V are combined by a post-processing function C^{POST} for generating the new chaining value W. Given M and V, the output W of the compression function can thus be written as

$$W = C^{\mathrm{POST}}(M, V, F(C^{\mathrm{PRE}}(M, V)) .$$

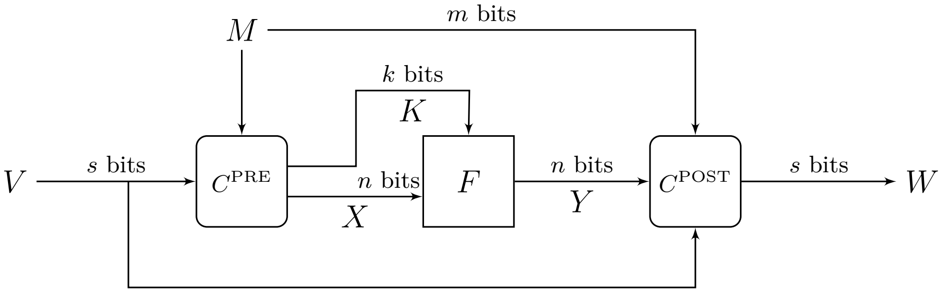


Fig. 1. General compression function of $(m + s)$ to s bits. In [32], F is a blockcipher with a k-bit key, while in [6] F is an unkeyed permutation with $n = s$.

Here, we consider the general compression function proposed by Stam where we suppose that F can also be a permutation, *i.e.*, it accommodates $k = 0$. In other words, this description also includes the compression function of any parazoa [6]. The sizes s, m, and n can then take any non-zero value and k can take any value including 0. This differs significantly from the case of parazoas where $n = s$. Then, a lot of different modes of operation fit this generalized compression function. For instance, in the sponge construction, where C^{PRE} xors M to a part of V, we have $s = n$, $k = 0$, and $C^{\mathrm{POST}}(M, V, Y) = Y$. Several types of C^{PRE} and C^{POST} used in the main hash function constructions are depicted in Tables 3 and 4, Page 15.

Within the indifferentiability framework, the function F is considered ideal, implying that the main differences in the security levels achieved by the variants on this broadened mode of operation will originate from the C^{PRE} and the C^{POST} function.

2.2 Iterating the broadened Stam compression function

We now describe a framework that enables to build hash functions based on the broadened Stam compression function. Our goal is to encompass as many real-life hash functions as possible, especially the SHA-3 semi-finalists. We can notice that most of them rely on the use of a unique internal primitive, which is compatible with the compression model we are considering, and iterate the compression function on a sequence of message blocks. The digest value is then a part of the output of the last compression function. For these reasons, we model the analyzed hash functions by the Chop-MD construction [21], which is defined as follows.

Let pad be a generic padding function, $\mathsf{pad} : \{0,1\}^* \to (\{0,1\}^m)^*$, which computes injectively a sequence of fixed-length blocks from a bitstring of arbitrary length, and a compression function $\Gamma : \{0,1\}^{s+m} \to \{0,1\}^s$. For a given message M such that $\mathsf{pad}(M) = M_1||\ldots||M_k$, the hash computation consists in iterating the compression function on the sequence of message blocks, updating in the process a chaining variable V. This chaining variable is initialized by a constant value V_0 and takes intermediate values V_i such that

$$\forall i \in \{1, \ldots, k\}, V_i = \Gamma(V_{i-1}, M_i) \; .$$

The digest of M is then defined as the first ℓ_h bits of V_k, denoted $V_k|_{\ell_h}$. The next section shows that the general form of the compression function combined with an appropriate padding provides a large amount of latitude to include all features of most modes of operation, including the existence of a block counter or of final rounds.

2.3 Identifying SHA-3 candidates with the broadened mode of operation

To show that the broadened generic compression function representation is indeed very generic, we have studied and represented most of the SHA-3 second round candidates under this form. In Table 1 we can see the classification of the 9 hash functions that we could represent this way.

SHA-3 candidate	m	s	n	k	C^{PRE}	C^{POST}
BLAKE	512/1024	$\frac{m}{2}$	m	m	$K = M, X = Exp(V)$	$W = Comp(Y) \oplus V$
JH	512	1024	1024	0	$X = Exp(M) \oplus V$	$W = Y \oplus Exp(M)$
KECCAK	1088/576	1600	1600	0	$X = Exp(M) \oplus V$	$W = Y$
Skein	512	512	512	512	$K = M, X = V$	$W = Y \oplus M$
Cubehash	8	1024	1024	0	$X = Exp(M) \oplus V$	$W = Y$
ECHO	1536/1024	512/1024	2048	0	$X = V\|\|M$	$W = Comp((M\|\|V) \oplus Y)$
Hamsi	32	ℓ_h	$2\ell_h$	0	$X = V\|\|Exp(M)$	$W = Comp(Y) \oplus V$
Shabal	512	1408	896	1024	$K = C\|\|M, X = (B + M)\|\|A$	$W = Y\|\|(Comp(V) - M)$
Shavite-3	512/1024	$\frac{m}{2}$	$\frac{m}{2}$	m	$K = M, X = V$	$W = Y \oplus V$

Table 1. Identifying the second-round SHA-3 candidates as broadened general compression functions. The functions *Exp* (resp. *Comp*) are simple linear expansion functions (resp. linear compression functions).

In addition, we provide in Table 5 (Page 16) the information about their finalization functions and padding rules. Since our analysis considers all these functions as Chop-MD based functions, where finalization is just truncation, we need to provide an alternative description of some of them. In this new description, counters and final rounds will be taken into account as parts of the message padding and of C^{PRE} and C^{POST}. For instance, the existence of a block counter added to the chaining variable can be represented by larger message blocks (including both the message block and the counter). But, while the counter is usually included in the inputs of the block cipher (see e.g. [10]), here the insertion of the counter is handled by C^{PRE} only, which leads to a more realistic model in many cases. The final rounds are also considered as an additional padding after the usual one, with the appropriate number of zeroes (or repetitions of the last message block in the case of Shabal) for covering all the final rounds (see Appendix A for details).

Functions that do not fit the model. There are some of the second round functions that we have not been able to include in this scenario. This is mainly due to the fact that they use in parallel several similar internal functions or permutations. Then, since the model considers a single ideal block cipher, the other ones should be included in the definition of C^{PRE} or of C^{POST}, making the analysis not directly applicable with our method. For instance, in the case of Grøstl, two similar permutations P and Q are applied in parallel. If we consider one of them as the central ideal permutation, the other one should be included in the C^{POST} block. Luffa also uses several permutations in parallel for each iteration of the compression function, whereas the construction we study only allows for one permutation or block cipher computation per compression function. Another example of function that does not fit our model for the same reason is the RIPEMD family.

The compression function of the SHA-3 semi-finalist BMW uses a different compression function after the last iteration. SIMD uses a postprocessing function that uses operations that are similar to the ones used by the main primitive. Therefore, it seems unclear whether the postprocessing function can be considered as deterministic, whereas the main primitive is modeled as an idealized block cipher. Finally, Fugue uses an internal primitive that operates only on a small part of the chaining variable. Therefore, its internal primitive cannot be considered ideal. A similar argument applies to other hash functions that were designed before the SHA-3 competition, like RadioGatún or Panama.

3 Indifferentiability proof using a generic sequence of games

In this section, we use the game-hopping technique to determine an indifferentiability bound for the previously described iterated broadened mode of operation. As the security of the construction depends on C^{PRE} and C^{POST} functions, this bound takes into account the properties of these functions, through quantities defined in Section 3.3. In Section 4, we evaluate these quantities for the choices of C^{PRE} and C^{POST} functions of SHA-3 candidates whose domain extenders fit into the general representation introduced in Section 2 and deduce indifferentiability bounds for each of them.

3.1 The indifferentiability framework

The indifferentiability framework has been introduced by Maurer *et al.* [28] as a generic tool to study the security of cryptosystems, and its application to the field of hash functions has been first proposed by Coron *et al.* [21]. In the case of hash functions, it enables to study the security of a domain extender $\mathcal{C}$. Informally, an indifferentiability proof guarantees that a hash function constructed by combining $\mathcal{C}$ with an ideal primitive is as secure as a random oracle. A more recent result by Ristenpart *et al.* [31] clarifies which security notions are covered by the indifferentiability framework. Namely, only the security against one-stage adversaries is guaranteed by an indifferentiability proof.

The concept of indifferentiability relies on a security game played between a distinguisher $\mathcal{D}$ and an oracle system. $\mathcal{D}$ interacts either with a hash function obtained by running a domain extender $\mathcal{C}$ implemented with a fixed-input-length random oracle (FIL-RO) $\mathcal{F}$ as its internal primitive, or with a variable-input-length random oracle (VIL-RO) $\mathcal{H}$.

The oracle system Σ based on $\mathcal{C}$ includes a direct access to the hash function $\mathcal{C}^{\mathcal{F}}$ and an access to the internal primitive $\mathcal{F}$. Some refinements introduced in [25, 24] may also give the adversary the possibility to query the inverse of $\mathcal{F}$, when $\mathcal{F}$ is a (possibly keyed) permutation. In the following, we take account of these refinements. $\mathcal{F}$ is therefore modeled as an ideal cipher $\mathcal{E}$ (or as a random permutation if $k = 0$) instead of a FIL-RO, and $\mathcal{D}$ can send queries to $\mathcal{E}$ and $\mathcal{E}^{-1}$.

The oracle system Σ' based on the VIL-RO $\mathcal{H}$ must give $\mathcal{D}$ a similar interface. Hash queries are sent directly to $\mathcal{H}$, and queries to the internal primitive or its inverse are dealt with by a specific Turing machine called *simulator* and denoted $\mathcal{S}$. To build its answers, $\mathcal{S}$ also has an access to $\mathcal{H}$. "Hash queries", or H-queries, refers to queries sent either to $\mathcal{C}^{\mathcal{F}}$ or $\mathcal{H}$. "Primitive queries", or F-queries, refers to queries sent either to $\mathcal{F}$ or $\mathcal{S}^{\mathcal{H}}$, for an application of the primitive or its inverse.

The indifferentiability framework is summarized on Figure 2. The distinguisher tries to determine whether he is interacting with Σ or Σ' and returns accordingly a binary value.

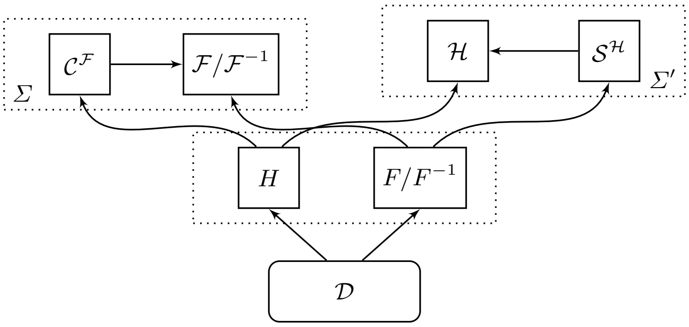

$\mathcal{C}^{\mathcal{F}}$
$\mathcal{F}/\mathcal{F}^{-1}$
Σ
$\mathcal{H}$
$\mathcal{S}^{\mathcal{H}}$
Σ'
H
F/F^{-1}
$\mathcal{D}$

Fig. 2. Indifferentiability of a domain extender $\mathcal{C}$ from a random oracle $\mathcal{H}$

The advantage of a given distinguisher is then defined as:

$$\mathsf{Adv}(\mathcal{D}) = \left|\Pr\left[\mathcal{D}^{\Sigma} = 1\right] - \Pr\left[\mathcal{D}^{\Sigma'} = 1\right]\right| .$$

We define the indifferentiability of $\mathcal{C}^{\mathcal{F}}$ from $\mathcal{H}$ as follows.

Definition 1. *$\mathcal{C}^{\mathcal{F}}$ is $(\varepsilon, N_{\mathcal{D}}, N_{\mathcal{S}})$-indifferentiable from $\mathcal{H}$ if there exists a simulator $\mathcal{S}$ making at most $N_{\mathcal{S}}$ queries to $\mathcal{H}$ such that the advantage of any distinguisher $\mathcal{D}$ making H- and F-queries of total weight at most $N_{\mathcal{D}}$ satisfies $\mathsf{Adv}(\mathcal{D}) \leq \varepsilon$. The total weight of queries is the sum of the blocklengths of all H-queries and of the number of F-queries.*

3.2 Design principles of the simulator and of the new sequence of games

We now apply this framework to the iterated broadened mode of operation. Let $\mathcal{C}$ denote this domain extension, $\overline{\mathcal{C}}$ this same extension without the truncation of the final chaining value.

First, we prove that the indifferentiability bound only depends on the image set of the padding function pad, and on the set of all prefixes of elements in this image set.

Lemma 1 (Domain modification). *Let π be a bijection from E to F such that π and π^{-1} can be easily computed, and let G be a finite set. We consider a construction $\mathcal{C}$ based on an internal primitive $\mathcal{P}$ such that $\mathcal{C}^{\mathcal{P}} : F \to G$.*

Let $\mathcal{H} : F \to G$ and $\mathcal{H}' : E \to G$ be two random oracles. Then, $\mathcal{C}^{\mathcal{P}}$ is $(\varepsilon, N_{\mathcal{D}}, N_{\mathcal{S}})$-indifferentiable from $\mathcal{H}$ if and only if $\mathcal{C}^{\mathcal{P}} \circ \pi$ is $(\varepsilon, N_{\mathcal{D}}, N_{\mathcal{S}})$-indifferentiable from $\mathcal{H}'$.

Proof. Suppose that $\mathcal{C}^{\mathcal{P}}$ is $(\varepsilon, N_{\mathcal{D}}, N_{\mathcal{S}})$-indifferentiable from $\mathcal{H}$, and let $\mathcal{S}$ be the simulator derived from the definition. We modify $\mathcal{S}$ to apply π to each hash query to obtain a simulator $\mathcal{S}'$. Now, we consider the oracle systems $(\mathcal{C}^{\mathcal{F}} \circ \pi, \mathcal{F})$ and $(\mathcal{H}', \mathcal{S}'^{\mathcal{H}'})$, and a distinguisher $\mathcal{D}'$ against the indifferentiability game defined by these systems. Then it can be easily shown that the distinguisher $\mathcal{D}$ obtained by applying π^{-1} to each hash query issued by $\mathcal{D}'$ has the same advantage against systems $(\mathcal{C}^{\mathcal{F}}, \mathcal{F})$ and $(\mathcal{H}, \mathcal{S}'^{\mathcal{H}})$ than $\mathcal{D}'$. Then, $\mathsf{Adv}(\mathcal{D}') \leq \varepsilon$. Therefore, $\mathcal{C}^{\mathcal{P}} \circ \pi$ is $(\varepsilon, N_{\mathcal{D}}, N_{\mathcal{S}})$-indifferentiable from $\mathcal{H}'$. The reverse implication is obtained by symmetry.

Let pad be an injective padding function, and S_{pad} be the set of its images. By applying the preceding lemma with $E = \{0,1\}^*$, $F = S_{\mathsf{pad}}$ and $\pi = \mathsf{pad}$, we deduce that the indifferentiability bounds for $\mathcal{C}^{\mathcal{F}}$ and $\mathcal{C}^{\mathcal{F}} \circ \mathsf{pad}$ from random oracles with the same domain are equal.

As a consequence, we can study the indifferentiability of a given construction in the first scenario. To take account of the influence of the padding, we use the set S_{pad} defined above and the set of prefixes of padded messages

$$S_{\mathsf{prefix}} = M \in (\{0,1\}^m)^* \text{ s.t. } \exists M' \in (\{0,1\}^m)^+, M||M' \in S_{\mathsf{pad}} \ .$$

Before giving our main indifferentiability result in Theorem 1, we summarize how we construct our simulator and bound the advantage of a potential distinguisher. The proof is based on a sequence of games: we build our final simulator as a sequence of elementary modifications starting from a passive simulator.

Simulation of $\mathcal{F}$. $\mathcal{S}$ has to simulate the answers to F-queries. It should ensure that the returned values are distributed according to a uniform choice of $\mathcal{F}$, and that applying the construction $\mathcal{C}$ to the answers for a given message M leads to the same results as queries to the random oracle $\mathcal{H}$. To this end it maintains two lists during the game. The first one is the list $\mathcal{L}_{\mathsf{values}} = \{(X, K, Y) \text{ s.t. } \mathcal{F}(K, X) = Y\}$ of the values of $\mathcal{F}$ that have already been defined. The second one is the list of identified intermediate values of the chaining variable $\mathcal{L}_{\mathsf{prefix}} = \{(V, \mu)\}$. The pair (V, μ) is in this list if μ is in S_{prefix} and V is the chaining variable obtained by applying the construction $\overline{\mathcal{C}^{\mathcal{F}}}$ to μ.

Our simulator is depicted on Figure 8. Informally, it works as follows. When it receives an F-query (K, X) to the internal primitive, $\mathcal{S}$ has to detect if the computation of $F(K, X)$ can be interpreted as the last iteration of the compression function in the computation of a digest $H(M)$: in this case, the returned value should be consistent with the random oracle. To identify a query (K, X) as the next step in a hash computation, $\mathcal{S}$ uses the following subroutine :

Prefix_identify :

$$(\mathcal{L}_{\mathsf{prefix}}, K, X) \mapsto \begin{cases} (V, \mu, M) \text{ s.t. } (V, \mu) \in \mathcal{L}_{\mathsf{prefix}},\ \mu||M \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}},\ C^{\mathrm{PRE}}(V, M) = (X, K) \\ \perp, \text{ if such } (V, \mu) \text{ does not exist.} \end{cases}$$

If this subroutine returns (V, μ, M) such that $\mu||M \in S_{\mathsf{pad}}$, the F-query is the last step of the computation of the digest $H(\mu||M)$ and $\mathcal{S}$ has to return an answer that is both consistent with $\mathcal{H}$ and uniformly distributed (like the answers of the $\mathcal{F}$ oracle). As a consequence, we require that, for any fixed chaining variable V and message block M, the first ℓ_h bits of the output of $Y \mapsto C^{\mathrm{POST}}(M, V, Y)$ take all possible values in $\{0,1\}^{\ell_h}$. This assumption holds for all C^{POST} identified in Table 4, and is less restrictive that the one in [6] which requires that $Y \mapsto C^{\mathrm{POST}}(M, V, Y)$ is bijective. Then, $\mathcal{S}$ queries $\mathcal{H}$ to get $\mathcal{H}(\mu||M)$.

Moreover, we suppose that there exists a sampling algorithm Samp that generates the value Y consistently with the answers of the random oracle. More precisely, *samp* depends on C^{POST} such that $\mathsf{Samp}(M, V, h, \mathcal{L}_{\mathsf{values}})$ and has to return $Y \in \{0,1\}^n$ satisfying:

- $C^{\mathrm{POST}}(M, V, Y)|_{\ell_h} = h$;
- $\forall (M, V)$, $\sum_{Y\in\{0,1\}^n} \left| 2^{-\ell_h} \Pr\left[\mathsf{Samp}(M, V, h, \mathcal{L}_{\mathsf{values}}) = Y\right] - \Pr\left[U(\mathcal{L}_{\mathsf{values}}, M, V) = Y\right] \right| \leq \Delta$.

In the last formula, Δ is a constant bound and $U(\mathcal{L}_{\mathsf{values}}, M, V)$ is the uniform distribution over the set of images Y that do not already appear in $\mathcal{L}_{\mathsf{values}}$ with the key K resulting from the application of C^{PRE} to (M, V). h denotes $C^{\mathrm{POST}}(M, V, Y)[_{\ell_h}$, and the factor $2^{-\ell_h}$ represents the probability that $\mathcal{H}(\mu||M)$ outputs h. We allow for a small bias on the definition of Y as a perfect simulation cannot always be achieved: as F is a permutation when the key is fixed, the output distribution of F is biased when some values are already defined with the same key.

$\mathcal{S}$ then uses Samp to generate its answer from (M, V) and $\mathcal{H}(\mu||M)$.

When it receives another F-query to the internal primitive (resp. its inverse), $\mathcal{S}$ selects the answer uniformly over the values that do not already have a preimage (resp. an image) by $\mathcal{F}$ under key K.

Bad events. The events that might prevent the simulator to maintain the consistency between $\mathcal{H}$ and $\mathcal{C}^{\mathcal{S}}$ are the following.

- If the simulation of a given output $F(K, X)$ is involved in the computation of the digests of two messages with distinct prefixes and common suffixes, the collision might propagate to the digests when applying $\mathcal{C}^{\mathcal{S}}$, whereas $\mathcal{H}$ ignores this relation.
- If the simulator returns a value X to a fresh query $F^{-1}(K, Y)$, and the computation of $F(K, X)$ is involved in the computation of the digest of a given message M, a constraint on the digest value may appear, which will be fulfilled by $\mathcal{C}^{\mathcal{S}}$ but not $\mathcal{H}$.
- If the simulator adds an element (V, μ) to $\mathcal{L}_{\mathsf{prefix}}$ after an F-query, and there is already a triple (X, K, Y) in $\mathcal{L}_{\mathsf{values}}$ and a message block M such that $C^{\mathrm{PRE}}(V, M) = (K, X)$, it might define the digest of any message beginning with $\mu||M$. This last case encompasses several attack strategies, including the length-extension attack.

About the internal memory of the simulator. In cryptography, constructive indifferentiability proofs involve

1. the description of a Turing machine that simulates the behaviour of an ideal primitive;
2. the evaluation of an upper bound of the advantage of the adversary.

The latter point is often achieved by defining bad events characterised by some property of the internal memory of the simulator. This gives an additional role to the simulator and its intermediate versions. Nevertheless, the identification of bad events does not necessarily involve a modification of the view of $\mathcal{D}$.

This method has the following drawback. Some unwanted event might occur due to H-queries, to which the final simulator does not have access. Therefore, the sequence of games usually involves an artificial access to H-queries for the simulator, which has then to be removed. This leads to two technical difficulties. First, this removal has to be done very carefully, as it modifies the internal memory of $\mathcal{S}$. This point is completely overlooked in some other proofs [21, 9, 6]: the difference between answers to H-queries is taken into account as their generations involve different processes, but the answers to F-queries are supposed to be produced in an equivalent way. We argue that this is something highly non-trivial, as the choice of a generation method might depend on the internal memory of $\mathcal{S}$. Nevertheless, our analysis tends to show that this shortcut does not lead to a mistake on the overall security bounds (see e.g. [27, 23] for details).

The other difficulty is that the usual method can lead to a more complex sequence of games. Because of the existence of several consecutive gaps on the view of the adversary, the indifferentiability bound is then multiplied by a constant but unnecessary factor.

Introducing the interceptor. In our proof, we build a second program called interceptor and denoted $\mathcal{I}$ that is used in intermediate games. Its role is to passively log all the H-queries, to translate them into sequences of F-queries by applying the construction $\mathcal{C}$ and to send these resulting queries to $\mathcal{S}$. We use two different versions of $\mathcal{I}$. When it is introduced in the sequence of games, $\mathcal{I}$ calls $\mathcal{S}$ as soon as it receives H-queries. In this way, the internal memory of $\mathcal{S}$ contains the knowledge of $\mathcal{D}$, which makes it easier to evaluate the probability that a bad event occurs. Later in the sequence of games, the second version of $\mathcal{I}$ delays calls to $\mathcal{S}$ to the end of the game, *i.e.* after all interactions involving $\mathcal{D}$. This guarantees that its action does not affect the view of the adversary.

Unicity of the simulation. Our proof is based on an 8-game sequence, labeled 0 to 7. Nevertheless, we describe only one method to generate the answers to F-queries, using the access to a random oracle. This is done between Games 5 and 6. The last transition consists in removing the detection of bad events. Transitions between Games 0 and 5 involve the introduction of bad events and modifications of the game scenarios that are depicted on Figure 3.

Bounding the advantage of $\mathcal{D}$. Our strategy to bound the advantage of $\mathcal{D}$ is the following. From Game 0 to Game 5, $\mathcal{I}$ and $\mathcal{S}$ identify unwanted events on the sequence of calls, but we show that they do not affect the view of $\mathcal{D}$. More precisely, in all these games, answers returned to $\mathcal{D}$ are always the values of $\mathcal{F}$, $\mathcal{F}^{-1}$ and $\mathcal{C}^{\mathcal{F}}$, where $\mathcal{F}$ is the ideal primitive that $\mathcal{S}$ does not control. We then need to bound the gap between Games 5 and 6, which is done by bounding the probability that a bad event occurs. Keep in mind that in these games, $\mathcal{I}$ calls $\mathcal{S}$ after all interactions with $\mathcal{D}$, so its action does not affect the view of $\mathcal{D}$. In Game 5, the occurrence of a bad event is fully determined by the random choices of $\mathcal{D}$ and the choice of an instance of $\mathcal{F}$. We define our bad events so they occur on the same random choices in Games 4 and 5. In Game 4, $\mathcal{S}$ has a real-time access to H-queries, therefore it is easier to bound the occurrence probability of a bad event.

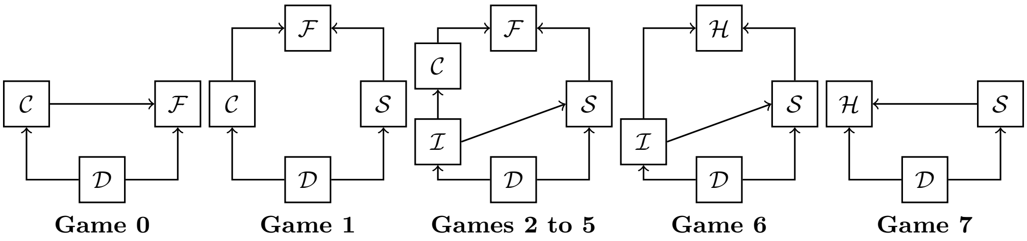


Fig. 3. Sequence of games used to prove Theorem 1.

Summary of the sequence of games. Overall, our sequence of games can be summarized as follows.

Game 0 We start with the original game taken from the definition of indifferentiability. In this game, $\mathcal{D}$ interacts with a system $\Sigma = (\mathcal{C}^{\mathcal{F}}, \mathcal{F})$, where $\mathcal{C}^{\mathcal{F}}$ answers only if the queried message belongs to S_{pad}.

Game 1 A passive simulator $\mathcal{S}_{1,2}$ is inserted in the game. It forwards F-queries to the ideal cipher $\mathcal{F}$, so that this modification does not affect $\mathcal{D}$'s view.

Games 2 to 4 The interceptor $\mathcal{I}_{2,3,4}$ is added to the game. It translates H-queries for the simulator and runs $\mathcal{S}$ immediately. Bad events detection is added in Game 4.

Game 5 This game is very similar to Game 4, except that $\mathcal{I}_{5,6}$ runs the simulator only at the end of the game.

Game 6 The oracle implementing the internal primitive $\mathcal{F}$ is replaced by the hash oracle $\mathcal{H}$. The construction $\mathcal{C}$ is withdrawn from the game. Therefore, $\mathcal{S}_6$ has to generate answers to F-queries itself. For each fresh query, if the corresponding computation can be interpreted as the last compression evaluation of a hash computation, $\mathcal{S}_6$ queries $\mathcal{H}$ to maintain consistency between the answers of both oracles. Otherwise, it generates a random value (with the restriction that $\mathcal{F}$ and $\mathcal{F}^{-1}$ must be bijective).

Game 7 The interceptor is withdrawn from the game and the bad event flags are withdrawn from the simulator. This does not modify $\mathcal{D}$'s view. The resulting game is the final game of the definition of indifferentiability, where $\mathcal{D}$ interacts with $\Sigma' = (\mathcal{H}, \mathcal{S}^{\mathcal{H}})$.

3.3 Indifferentiability result

The bounds obtained in [32, 6] on the complexity of collision attacks apply only if C^{PRE} and C^{POST} satisfy certain properties, for instance, if some related functions are invertible. Instead, we give an indifferentiability proof in the general case and bound the advantage of the adversary using the following quantities which depend on the properties of C^{PRE} and C^{POST}. We are thus able to provide a relevant bound in more cases. To illustrate what these quantities represent, we detail their values for two widely used constructions: the sponge functions with $\ell_h = m$ (and where the block is xored to the part of the state that corresponds to the digest), and Chop-MD with a Davies-Meyer compression function (DM Chop-MD).

Collisions attacks. The quantity λ_{Coll} appears in the occurrence probability of a local collision, *i.e.* an evaluation of F on a given value (K, X) that is part of two hash computations of messages with different prefixes. We use the following definition

$$\lambda_{\mathsf{Coll}} = \max_{V,\mu,M,V',\mu'} \left| \left\{ \begin{array}{r} Y \text{ s.t. } \exists M^*, M', V^*,\ C^{\mathrm{POST}}(V,M,Y) = V^*, C^{\mathrm{PRE}}(V',M') = C^{\mathrm{PRE}}(V^*,M^*) \\ \text{and } \mu||M||M^* \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}},\ \ \mu'||M' \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}} \end{array} \right\} \right| .$$

In the case of parazoas, including sponge functions, $Y \mapsto C^{\mathrm{POST}}(V, M, Y)$ is bijective. Then, we always have $V^* = Y$, and we can have $C^{\mathrm{PRE}}(V', M') = C^{\mathrm{PRE}}(Y, M)$ if V' and Y collide on $n - m$ bits, which leaves 2^m possibilities for Y. Thus $\lambda_{\mathsf{Coll}} = 2^m$.

Another situation is the case where C^{PRE} is injective. Then, λ_{Coll} is the maximal number of Y such that $C^{\mathrm{POST}}(V, M, Y) = V'$ for a given (V, V', M) which is valid with respect to the padding function. For example, in the case of DM Chop-MD, there is a single choice for Y, and $\lambda_{\mathsf{Coll}} = 1$.

Meet-in-the-middle attacks. The quantity λ_{Mitm} is used to bound the probability of a meet-in-the-middle distinguishing attack.

$$\lambda_{\mathsf{Mitm}} = \max_{V,\mu,K} \left| \left\{ X \text{ s.t. } \exists M,\ \ C^{\mathrm{PRE}}(V, M) = (K, X) \text{ and } \mu||M \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}} \right\} \right| .$$

For sponges, if V collides with X on the last 2^{n-m} bits, an accurate choice of M leads to $V \oplus (M||0) = X$. Therefore, $\lambda_{\mathsf{Mitm}} = 2^m$. In the case of DM Chop-MD, $X = V$ and $\lambda_{\mathsf{Mitm}} = 1$.

Length-extension attacks. We now define two quantities related to the probability of length-extension attacks: $\lambda_{\mathsf{Extension},1}$ corresponds to the maximum number of Y which, when combined to the same pair (V, M), lead to the same value after applying $C^{\mathrm{PRE}} \circ C^{\mathrm{POST}}$. The other quantity $\lambda_{\mathsf{Extension},2}$ corresponds to the cardinality of the same set of values Y with the additional constraints that the first ℓ_h bits of the output of C^{POST} are fixed.

$$\lambda_{\mathsf{Extension},1} = \max_{X,K,V,\mu,M} \left|\left\{Y \text{ s.t. } \exists M', C^{\mathrm{PRE}}(C^{\mathrm{POST}}(V,M,Y),M') = (K,X), \mu||M||M' \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}}\right\}\right| ,$$

$$\lambda_{\mathsf{Extension},2} = \max_{X,K,V,\mu,M,h} \left|\left\{\begin{array}{c} Y \text{ s.t. } \exists M',\ C^{\mathrm{POST}}(V,M,Y)|_{\ell_h} = h,\ C^{\mathrm{PRE}}(C^{\mathrm{POST}}(V,M,Y),M') = (K,X) \\ \text{and } \mu||M||M' \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}} \end{array}\right\}\right| .$$

These constants bound the number of values Y that can lead to a guess (K, X) made by $\mathcal{D}$ when $\mathcal{D}$ knows the H-part of the intermediate chaining variable and when he does not. In the case of the DM Chop-MD, only one value of Y can lead to X if V is fixed. Therefore, $\lambda_{\mathsf{Extension},1} = \lambda_{\mathsf{Extension},2} = 1$. In the case of the sponge construction, when h is not fixed, 2^m values of Y can lead to a given X ($\lambda_{\mathsf{Extension},1} = 2^m$), but given h and X, only 1 of the values works, and $\lambda_{\mathsf{Extension},2} = 1$.

λ_{Pre}. To estimate the resistance to length-extension attacks, we need to know, given a guess (K, X) by the adversary, how many previously obtained distinct hashes h_i can be reached by an appropriate choice of v_i, M_i and the application of C^{PRE} to $(h_i||v_i, M_i)$. Therefore, we denote $S(K, X)$ the set $\{h|\exists v, M, C^{\mathrm{PRE}}(h||v, M) = (K, X)\}$. To establish a bound, we can suppose that these sets for different values of (K, X) are either identical or disjoint and we denote by λ_{Pre} their maximal size.

However, we could also wish to relax this constraint. We consider a partition of $\{0, 1\}^{\ell_h}$, $(S_1, \ldots, S_t)$ such that each $S(K, X)$ is included in one of the S_t's. Instead of being either disjoint or identical, the sets $S(K, X)$ are now either disjoint or included in the same set of a partition. We then denote λ_{Pre} the maximal size reached by one of the S_t's.

In the sponge case, any hashed output can be completed into V combined with a given message block to lead to any X. Therefore, $\lambda_{\mathsf{Pre}} = 2^{\ell_h}$. For DM Chop-MD, the hash output is a part of the input of the next iteration of F. Thus, $\lambda_{\mathsf{Pre}} = 1$.

μ_{Post}. We also define a value that is only related to C^{POST}, which is the minimal number of preimages which can be obtained for $Y \mapsto C^{\mathrm{POST}}(V, M, Y)|_{\ell_h}$ for any fixed (V, M):

$$\mu_{\mathsf{Post}} = \min_{M,V,h} \left|\{Y \text{ s.t. } C^{\mathrm{POST}}(V, M, Y)|_{\ell_h} = h\}\right| .$$

This constant gives a lower bound on the number of possible Y that map to a given hash output h, for fixed (V, M) entering C^{POST}. In most common designs, this quantity is constant. It seems clear that a uniform distribution for Y should lead to a uniform distribution for h, which is the case of the sponge construction and DM Chop-MD. In both cases, this leads to $\mu_{\mathsf{Post}} = 2^{n-\ell_h}$.

λ_{Post}. Another possibility for $\mathcal{D}$ is to try to exploit a length-extension-like property by calling F^{-1}. To bound the success probability of such a strategy, we define the sets $S'(V, M, h)$ as $\{Y|C^{\mathrm{POST}}(V, M, Y)|\ell_h = h\}$. We also suppose that for distinct values of (V, M, h), these sets are either disjoint or identical, and bound their size by λ_{Post}. As for the case of λ_{Pre}, we can relax this constraint by defining a partition, each of the $S'(V, M)$ being included in one of the sets of the partition. In most applications, the sets of values Y that lead to a common h value do not depend on V and M and $\lambda_{\mathsf{Post}} = 2^{n-\ell_h}$. This is the case for DM Chop-MD and the sponge construction.

To bound the probability of length-extension attacks, we need to evaluate the number of hash values that are in the same set S_i or S'_j during the game. This can be expressed with the function

$$g(N, \lambda, \ell) = \min(N, (1 + e)N\lambda 2^{-\ell} + 2\ell - 2\log_2(\lambda)) .$$

This quantity bounds the sum of the probabilities of occurrence of i-multicollisions, $0 \leq i \leq N$, in a set of N values drawn independently following a distribution such that each value has probability at most $\lambda/2^{\ell}$. We refer to Appendix B.2 for the proof of this bound and its relationship with length-extension attacks.

We can now give our indifferentiability bound.

Theorem 1. *Let us consider a domain extender $\mathcal{C}$, obtained by iterating a broadened Stam compression function, and a padding function* pad *with corresponding sets S_{pad} and S_{prefix}. Let $\mathcal{F}$ be an ideal cipher (or an ideal permutation if $k = 0$) and $\mathcal{H}$ be a random oracle with domain S_{pad}. Let $\mathcal{S}$ be the simulator defined on Figure 8 in Appendix B and let $\mathcal{D}$ be a distinguisher that issues H and F queries of total weight at most $N_{\mathcal{D}}$. Then, if $\mu_{\mathsf{Post}} > 1$,*

$$\mathsf{Adv}(\mathcal{D}) \leq 2N_{\mathcal{D}}\Delta + \frac{\max(\lambda_{\mathsf{Coll}}, \lambda_{\mathsf{Mitm}})N_{\mathcal{D}}^2}{2(2^n - N_{\mathcal{D}})}$$
$$+ \max\left(\frac{\lambda_{\mathsf{Extension},1}N_{\mathcal{D}}^2}{2(2^n - N_{\mathcal{D}})}, \frac{g(N_{\mathcal{D}}, \lambda_{\mathsf{Pre}}, \ell_h)\lambda_{\mathsf{Extension},2}N_{\mathcal{D}}}{\mu_{\mathsf{Post}} - N_{\mathcal{D}}}, \frac{g(N_{\mathcal{D}}, \lambda_{\mathsf{Post}}, n)N_{\mathcal{D}}}{\mu_{\mathsf{Post}} - N_{\mathcal{D}}}\right) .$$

where Δ is defined in Section 3.2. Furthermore, $\mathcal{S}$ issues at most $N_{\mathcal{D}}$ hash queries.

More specifically, if the padding function is prefix-free, we have

$$\mathsf{Adv}(\mathcal{D}) \leq 2N_{\mathcal{D}}\Delta + \frac{\max(\lambda_{\mathsf{Coll}}, \lambda_{\mathsf{Mitm}})N_{\mathcal{D}}^2}{2(2^n - N_{\mathcal{D}})} + \max\left(\frac{\lambda_{\mathsf{Extension},1}N_{\mathcal{D}}^2}{2(2^n - N_{\mathcal{D}})}, \frac{g(N_{\mathcal{D}}, \lambda_{\mathsf{Post}}, n)N_{\mathcal{D}}}{\mu_{\mathsf{Post}} - N_{\mathcal{D}}}\right) .$$

4 Application to several known constructions

One of the main interests of the previous result is that it enables us to determine indifferentiability bounds for domain extenders of concrete hash functions, including several of the SHA-3 candidates. For all functions we study, we can sample Y with the uniform distribution over the values that fulfill $C^{\mathrm{POST}}(V, M, Y)|_{\ell_h} = h$ when h is uniformly distributed. As at most $N_{\mathcal{D}}$ values for Y cannot be reached if F is a permutation, we have $\Delta = \frac{1}{2^n - N_{\mathcal{D}}}$.

As an illustration, we first apply the theorem to both previously considered examples: the sponge construction and Davies-Meyer Chop-MD. For the sponge construction, we get that

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{2N_{\mathcal{D}}}{2^n - N_{\mathcal{D}}} + \frac{2^{\ell_h}N_{\mathcal{D}}^2}{2(2^n - N_{\mathcal{D}})} + \frac{2N_{\mathcal{D}}^2}{2^{n-\ell_h} - N_{\mathcal{D}}} ,$$

which leads to the same bound as [9], with an additional (negligible) term corresponding to length-extension attacks. In the case of DM Chop-MD, we get

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{2N_{\mathcal{D}}}{2^n - N_{\mathcal{D}}} + \frac{N_{\mathcal{D}}^2}{2(2^n - N_{\mathcal{D}})} + \frac{2N_{\mathcal{D}}}{2^{n-\ell_h} - N_{\mathcal{D}}} \min\left(N_{\mathcal{D}}, (1+e)N_{\mathcal{D}}2^{-\ell_h} + 2\ell_h\right) .$$

The bound improves the result obtained in [21]. It is worth noticing that the best previously known indifferentiability bound for Chop-MD is due to Chang and Nandi [18] but it does not take into account the fact that the internal primitive is a permutation.

Another interesting construction is a variant of the Davies-Meyer mode based on a block cipher with blocksize twice larger than the chaining value, *i.e.*, with $s = \ell_h$ and $n = 2\ell_h$. Such a construction can use for instance $C^{\mathrm{PRE}}(M, v) = (M, Exp(V))$ and $C^{\mathrm{POST}}(M, V, Y) = Comp(Y) \oplus V$ as in the simplified mode of operation of BLAKE when the counter is omitted. In this case, we have $\lambda_{\mathsf{Coll}} = 2^{\ell_h}$, $\lambda_{\mathsf{Mitm}} = 1$ and $\lambda_{\mathsf{Extension},1} = \mu_{\mathsf{Post}} = \lambda_{\mathsf{Post}} = 2^{\ell_h}$. Then, for a prefix-free padding, we obtain

$$\mathsf{Adv}(\mathcal{D}) \leq \frac{2N_{\mathcal{D}}}{2^n - N_{\mathcal{D}}} + \frac{2^{\ell_h+1}N_{\mathcal{D}}^2}{2(2^n - N_{\mathcal{D}})} + \frac{2N_{\mathcal{D}}}{2^{n-\ell_h} - N_{\mathcal{D}}} \min\left(N_{\mathcal{D}}, (1+e)N_{\mathcal{D}}2^{-\ell_h} + 2\ell_h\right) .$$

Then, an interesting observation is that this bound is similar to the bound obtained for sponges with the same blocksize $n = 2\ell_h$ and digest size ℓ_h. But, the attacks meeting the bound are different: meet-in-the-middle attacks have complexity $2^{\frac{\ell}{2}}$ for sponges while they are much more expensive for the other construction. On the other hand, the prefix-free Davies-Meyer with double blocksize is vulnerable to collision attacks with complexity $2^{\frac{\ell_h}{2}}$. Also, it is worth noticing that this construction is indifferentiable up to $2^{\frac{\ell_h}{2}}$ queries while the chopped Davies-Meyer construction which is based on a similar block cipher is indifferentiable up to 2^{ℓ_h} queries.

Application to some SHA-3 candidates. The determination of the bounds λ_{Coll}, λ_{Mitm}, $\lambda_{\mathsf{Extension},1}$, $\lambda_{\mathsf{Extension},2}$, λ_{Post}, μ_{Post} and λ_{Pre} are pretty straightforward and summarized in Table 2 for some SHA-3 candidates, as well as the resulting indifferentiability bound for each of the functions. Our proof gives a significant bound for hash functions based on an ideal primitive whose output size is larger than the digest size. It is worth noticing that this is not restricted to wide-pipe designs since it also includes constructions using a chaining variable of the same size as the digest (for instance, BLAKE or Hamsi). For Hamsi, this proof is to our best knowledge the first one that takes into account the invertibility of the internal primitive.

SHA-3 candidate	λ_{Coll}	λ_{Mitm}	P-F	$\lambda_{\mathsf{Extension},2}$	$\lambda_{\mathsf{Extension},1}$	λ_{Post}	μ_{Post}	λ_{Pre}	Bound
BLAKE	2^{ℓ_h}	2	yes	n/a	2^{ℓ_h}	2^{ℓ_h}	2^{ℓ_h}	n/a	$\approx \frac{(3+2e)2^{\ell_h} N_{\mathcal{D}}^2}{2\times 2^{2\ell_h} - N_{\mathcal{D}}}$
JH	2^{512}	2^{512}	no	2^{512}	2^{512}	$2^{1024-\ell_h}$	$2^{1024-\ell_h}$	1	$\approx \frac{2\ell_h \times 2^{512} N_{\mathcal{D}}}{2^{1024-\ell_h} - N_{\mathcal{D}}}$
KECCAK-256	2^{1088}	2^{1088}	no	1	2^{1088}	2^{1344}	2^{1344}	2^{256}	$\approx \frac{2N_{\mathcal{D}}^2}{2^{512} - N_{\mathcal{D}}}$
KECCAK-512	2^{576}	2^{576}	no	2^{64}	2^{576}	2^{1088}	2^{1088}	2^{512}	$\approx \frac{2N_{\mathcal{D}}^2}{2^{1024} - N_{\mathcal{D}}}$
Cubehash	2^{9}	2^{9}	yes	n/a	2^{9}	$2^{1024-\ell_h}$	$2^{1024-\ell_h}$	n/a	$\approx \frac{(2^9+e+1)N_{\mathcal{D}}^2}{2^{1024} - N_{\mathcal{D}}}$
ECHO	$2^{2048-2\ell_h}$	1	yes	n/a	$2^{2048-2\ell_h}$	$2^{2048-\ell_h}$	$2^{2048-\ell_h}$	n/a	$\approx \frac{2^{2048-2\ell_h} N_{\mathcal{D}}^2}{2^{2048} - N_{\mathcal{D}}}$
Hamsi	2^{ℓ_h}	1	yes	n/a	2^{ℓ_h}	2^{ℓ_h}	2^{ℓ_h}	n/a	$\frac{2^{\ell_h} N_{\mathcal{D}}^2}{2^{2\ell_h} - N_{\mathcal{D}}}$
Shabal-512	1	1	yes	n/a	1	2^{384}	2^{384}	n/a	$\approx \frac{1024 N_{\mathcal{D}}}{2^{384} - N_{\mathcal{D}}}$
Shabal-256	1	1	yes	n/a	1	2^{640}	2^{640}	n/a	$\approx \frac{N_{\mathcal{D}}^2}{2^{896} - N_{\mathcal{D}}}$

Table 2. Indifferentiability bounds for the second-round SHA-3 candidates deduced from Theorem 1.

When the block size of the ideal primitive F is equal to the size of the digest, we have $\mu_{\mathsf{Post}} = 1$ and our bound does not apply. Furthermore, our simulator is vulnerable to the same reverse attacks as the simulators in [21, 19]: $\mathcal{D}$ queries $H(M)$ where M is a single block message, and computes Y such that $C^{\mathrm{POST}}(IV, M, Y) = h$ and $(K, X) = C^{\mathrm{PRE}}(V, M)$; then he queries $F^{-1}(K, Y)$. The answer should be X, that the simulator has no reason to know. To get a result on these functions, we have to modify the simulator as suggested in [27]: when receiving a request (K, Y) to F^{-1}, $\mathcal{S}$ simulates all the values $F(K, X)$ for which there is a (V, μ) in $\mathcal{L}_{\mathsf{prefix}}$ and a message block M such that $C^{\mathrm{PRE}}(V, M) = (K, X)$ and $\mu || M \in S_{\mathsf{pad}}$. However, this may highly increase the number of hash queries issued by the simulator, and the properties of the padding play then an important role. Finding a relevant unified indifferentiability bound is then an open issue in this case.

5 Conclusions

Our result provides an indifferentiability bound for many blockcipher-based or permutation-based hash functions, as soon as the block size of this internal primitive is greater than the digest size. An interesting observation is that the choice between an ideal keyed or an unkeyed permutation appears to have a little influence on the indifferentiability bound — but, an ideal keyed permutation is a much more complex primitive than an ideal permutation. For instance, our result allows to compare some constructions like a parazoa with blocksize $n = 2\ell_h$, a chopped Davies-Meyer construction with blocksize $n = 2\ell_h$ and a prefix-free Davies-Meyer construction with chaining value of size ℓ_h based on a blockcipher operating on blocks of size $2\ell_h$, like the BLAKE mode of operation.

Another important issue is that our result takes into account a realistic representation of most compression functions. For instance, it uses the fact that the compression function in BLAKE is based on an ideal block cipher with block size $2\ell_h$. The indifferentiability bound that we derive is then very different from the usual argument [5] which identifies BLAKE mode of operation with the Haifa construction iterating an ideal compression function from $(13\ell_h/4)$ to ℓ_h bits. Actually, it has been recently observed that, when the counter insertion is included within the compression function, the compression function can then be easily distinguished from an ideal function [20, 2]. These works point out that a relevant indifferentiability proof can only be obtained if the ideal component corresponds to the block cipher. In the case of BLAKE, our general result gives the same bound as the dedicated proof given in [20, 2]. Moreover, a better identification of the components of the compression function which are expected to have a random behaviour may help some extension of indifferentiability proofs aiming at determining how some structural properties of the inner block cipher or permutation weaken the overall security of the hash function. Besides its direct application, we believe that this synthetic view will help future research on modes of operation for hash functions.

References

1. Elena Andreeva, Andrey Bogdanov, Bart Mennink, Bart Preneel, and Christian Rechberger. On security arguments of the second round sha-3 candidates. *Int. J. Inf. Sec.*, 11(2):103–120, 2012.
2. Elena Andreeva, Atul Luykx, and Bart Mennink. Provable Security of BLAKE with Non-Ideal Compression Function. *IACR Cryptology ePrint Archive*, 2011:620, 2011.
3. Elena Andreeva, Bart Mennink, and Bart Preneel. Security reductions of the second round SHA-3 candidates. In Mike Burmester, Gene Tsudik, Spyros S. Magliveras, and Ivana Ilic, editors, *ISC*, volume 6531 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2010.
4. Elena Andreeva, Bart Mennink, and Bart Preneel. Security reductions of the second round SHA-3 candidates. Cryptology ePrint Archive, Report 2010/381, 2010. `http://eprint.iacr.org/`.
5. Elena Andreeva, Bart Mennink, and Bart Preneel. Security Reductions of the Second Round SHA-3 Candidates. In *ISC*, volume 6531 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2011.
6. Elena Andreeva, Bart Mennink, and Bart Preneel. The parazoa family: generalizing the sponge hash functions. *Int. J. Inf. Sec.*, 11(3):149–165, 2012.
7. Mihir Bellare, Tadayoshi Kohno, Stefan Lucks, Niels Ferguson, Bruce Schneier, Doug Whiting, Jon Callas, and Jesse Walker. Provable security support for the Skein hash family. `http://www.skein-hash.info/sites/default/files/skein-proofs.pdf`, 2009.
8. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions. *Ecrypt Hash Workshop*, Barcelona, Spain, May 2007. `http://sponge.noekeon.org/`.
9. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the indifferentiability of the sponge construction. In *Advances in Cryptology — EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008.
10. Rishiraj Bhattacharyya, Avradip Mandal, and Mridul Nandi. Indifferentiability Characterization of Hash Functions and Optimal Bounds of Popular Domain Extensions. In *INDOCRYPT*, volume 5922 of *Lecture Notes in Computer Science*, pages 199–218. Springer, 2009.
11. Rishiraj Bhattacharyya, Avradip Mandal, and Mridul Nandi. Security analysis of the mode of JH hash function. In Seokhie Hong and Tetsu Iwata, editors, *FSE*, volume 6147 of *Lecture Notes in Computer Science*, pages 168–191. Springer, 2010.
12. Eli Biham and Orr Dunkelman. A framework for iterative hash functions - HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007. `http://eprint.iacr.org/`.
13. John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from pgv. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002.
14. John Black, Phillip Rogaway, Thomas Shrimpton, and Martijn Stam. An analysis of the blockcipher-based hash functions from PGV. *J. Cryptology*, 23(4):519–545, 2010.
15. Charles Bouillaguet, Pierre-Alain Fouque, and Gaëtan Leurent. Security Analysis of SIMD. In *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 351–368. Springer, 2011.
16. E. Bresson, A. Canteaut, B. Chevallier-Mames, C. Clavier, T. Fuhr, A. Gouget, T. Icart, J.-F. Misarsky, M. Naya-Plasencia, P. Paillier, T. Pornin, J.-R. Reinhard, C. Thuillet, and M. Videau. Shabal, a submission to NIST'cryptographic hash algorithm competition. Submission to the NIST Hash competition, 2008.
17. E. Bresson, A. Canteaut, B. Chevallier-Mames, C. Clavier, T. Fuhr, A. Gouget, T. Icart, J.-F. Misarsky, M. Naya-Plasencia, P. Paillier, T. Pornin, J.-R. Reinhard, C. Thuillet, and M. Videau. Indifferentiability with Distinguishers: Why Shabal does not require ideal ciphers. IACR ePrint Archive: Report 2009/199, 2009.
18. D. Chang and M. Nandi. Improved indifferentiabily security analysis of chopMD hash function. In *Fast Software Encryption - FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 429–443. Springer, 2008.
19. Donghoon Chang, Sangjin Lee, Mridul Nandi, and Moti Yung. Indifferentiable Security Analysis of Popular Hash Functions with Prefix-Free Padding. In *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 283–298. Springer, 2006.

20. Donghoon Chang, Mridul Nandi, and Moti Yung. Indifferentiability of the hash algorithm blake. *IACR Cryptology ePrint Archive*, 2011:623, 2011.
21. J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In *Advances in Cryptology — CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, 2005.
22. I. Damgård. A design principle for hash functions. In *Advances in Cryptology — CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1989.
23. Marion Daubignard, Pierre-Alain Fouque, and Yassine Lakhnech. Generic indifferentiability proofs of hash designs. In *25th IEEE Computer Security Foundations Symposium*, 2012.
24. Jonathan J. Hoch and Adi Shamir. On the Strength of the Concatenated Hash Combiner When All the Hash Functions Are Weak. In *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 616–630. Springer, 2008.
25. Moses Liskov. Constructing an Ideal Hash Function from Weak Ideal Compression Functions. In *Selected Areas in Cryptography 2006*, volume 4356 of *Lecture Notes in Computer Science*, pages 358–375. Springer, 2007.
26. S. Lucks. A failure-friendly design principle for hash functions. In *Advances in Cryptology — ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 474–494. Springer, 2005.
27. Yiyuan Luo, Zheng Gong, Ming Duan, Bo Zhu, and Xuejia Lai. Revisiting the indifferentiability of pgv hash functions. Cryptology ePrint Archive, Report 2009/265, 2009.
28. U. Maurer, R. Renner, and C. Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In *Theory of cryptography – TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004.
29. R. Merkle. One way hash functions and DES. In *Advances in Cryptology — CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer, 1989.
30. Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer, 1993.
31. Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 487–506. Springer, 2011.
32. Martijn Stam. Blockcipher based hashing revisited. In *Fast Software Encryption - FSE 2009*, volume 5665 of *Lecture Notes in Computer Science*, pages 67–83. Springer, 2009.

A Examples of C^{PRE} and C^{POST} functions and finalization, counter and padding of the SHA-3 candidates

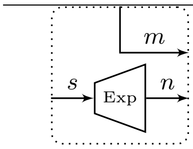
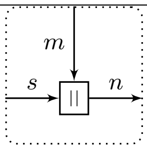
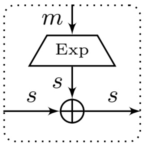
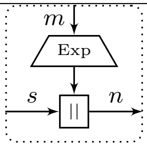
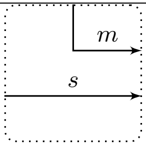
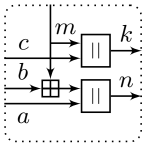
C^{PRE}	Parameters	C^{PRE}	Parameters
	$k = m$, $n > s$. $K = M, X = Exp(V)$		$k = 0$, $n = m + s$. $X = V\|\|M$
	$k = 0$, $n = s$, $m < s$. $X = Exp(M) \oplus V$		$k = 0$, $n > m + s$. $X = V\|\|Exp(M)$
	$k = m$, $n = s$. $K = M$, $X = V$		$s = a + b + c$, $m = b$, $k = m + c$, $n = a + b$. $K = C\|\|M$, $X = (B + M)\|\|A$

Table 3. Common types of C^{PRE}. The functions *Exp* are simple linear expansion functions, like for instance the concatenation with a constant. Symbol || denotes a concatenation.

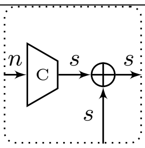
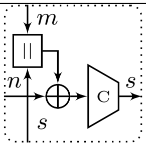
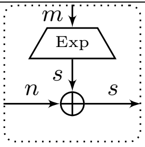
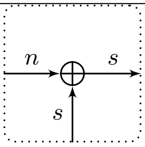
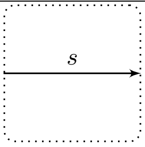
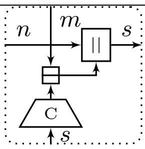
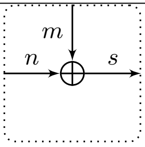
C^{POST}	Parameters	C^{POST}	Parameters
	$n > s$. $W = Comp(Y) \oplus V$		$n = s + m$. $W = Comp((M\|\|V) \oplus Y)$
	$m < n$, $n = s$. $W = Y \oplus Exp(M)$		$s = n$. $W = Y \oplus V$
	$n = s$. $W = Y$		$m + n = s$. $W = Y\|\|(Comp(V) - M)$
	$m = n = s$. $W = Y \oplus M$		

Table 4. Common types of C^{POST}. The functions *Comp* are simple linear compression functions, like fold or a truncation.

SHA-3 candidate	Finalization function	Padding	Counter
BLAKE	/	$m \leftarrow m\|\|10\dots01(\ell)_{64/128}$	64/128 bits
JH	truncate	$m \leftarrow m\|\|10\dots0(\ell)_{128}$	no
KECCAK	truncate	$m \leftarrow m\|\|10\dots01$	no
Skein	0-message block	$m \leftarrow m\|\|10\dots0$	block counter(64)
Cubehash	⊕1bit/80 rounds perm/truncate	$m \leftarrow m\|\|10\dots0$	no
ECHO	truncate	$m \leftarrow m\|\|10\dots0(\|h\|)_{16}(\ell)_{128}$	64 bits
Hamsi	last block: more rounds, ≠ constants	$m \leftarrow m\|\|10\dots0(\ell)_{64}$	no
Shabal	3 rounds with last block, same counter	$m \leftarrow m\|\|10\dots0$	block counter(64)
Shavite-3	truncated	$m \leftarrow m\|\|10\dots0(\ell)_{64}(\|h\|)_{16}$	64 bits

Table 5. Finalization functions, counter and padding of the SHA-3 candidates. We consider ℓ as the length of the message in bits before padding.

In order to include the features described in Table 5 within the chop-MD mode with a particular padding, we consider an equivalent description of the following hash functions:

- BLAKE, with message-block size $m = 576/1152$, as the counter is now included in the message block by the padding.
- Skein, with message-block size $m = 576$, as the counter is included by the padding, and the padding also adds an all-zero block at the end of the padded message.
- ECHO, with message-block size $m = 1600/1088$ (for the counter).
- Cubehash, with a new padding that adds 10 zero blocks at the end of the previous padded message, and where the message block has one additional bit, *i.e.*, $m = 9$, that is used for representing the xor of 1 before the final rounds (this bit is zero for all the blocks except the first of the last 10 ones).
- Shabal, with an increased block size of $m = 576$ (for the counter), and where the padding repeats the last block three times at the end.
- Shavite-3, with message-block size $m = 2 \times w + 64$ (for the counter).
- For Hamsi, the last iteration of F has more rounds and uses different constants. We consider the Hamsi mode of operation with an independent permutation for the last block. Therefore, the

message blocks contain one more bit, which is 1 only for the last block. C^{PRE} uses it to define a 1-bit key.

B Detailed indifferentiability proof of the broadened mode of operation

B.1 Detailed description of the different versions of the simulator

Game 0. $\mathcal{D}$ is interacting with $\Sigma = (\mathcal{C}^{\mathcal{F}}, \mathcal{F})$. No simulator is involved in this game.

```
Initialisation of $\mathcal{S}_{1,2}$

1. define $\mathcal{L}_{\mathsf{values}} = \emptyset$ and $\mathcal{L}_{\mathsf{prefix}} = \{(IV, \epsilon)\}$

Simulation of $F$
Input :     $(K, X)$
Output :    $Y$

1. call $\mathcal{F}$ to get $Y = F(K, X)$
2. add $(X, K, Y)$ to $\mathcal{L}_{\mathsf{values}}$
3. if $\mathsf{Prefix\_identify}(\mathcal{L}_{\mathsf{prefix}}, K, X) \neq \perp$
   (a) set $(V, \mu, M) = \mathsf{Prefix\_identify}(\mathcal{L}_{\mathsf{prefix}}, K, X)$
   (b) set $W = C^{\mathrm{POST}}(V, M, Y)$
   (c) if $\mu || M \in S_{\mathsf{prefix}}$, add $(W, \mu || M)$ to $\mathcal{L}_{\mathsf{prefix}}$
4. return $Y$

Simulation of $F^{-1}$
Input :     $(K, Y)$
Output :    $X$

1. call $\mathcal{F}^{-1}$ to get $X = F^{-1}(K, Y)$
2. add $(X, K, Y)$ to $\mathcal{L}_{\mathsf{values}}$
3. return $X$
```

Fig. 4. Simulator $\mathcal{S}_{1,2}$ for F and F^{-1} in Games 1 and 2.

Game 1. A passive simulator $\mathcal{S}_{1,2}$ (Figure 4) is introduced in Game 1. $\mathcal{S}$ receives queries to F or F^{-1} issued by $\mathcal{D}$, forwards them to $\mathcal{F}$, and forwards the answer to $\mathcal{D}$. It builds the list $\mathcal{L}_{\mathsf{values}} = \{(X, K, Y)\}$ by storing the queries and answers it receives. $\mathcal{S}_{1,2}$ also maintains the list $\mathcal{L}_{\mathsf{prefix}}$ of values of the chaining variable that can be identified with the intermediate value of a given hash computation. To achieve it, $\mathcal{L}_{\mathsf{prefix}}$ is initialized with $\{(IV, \epsilon)\}$, where ϵ denotes an empty sequence of message blocks. Then, $\mathcal{L}_{\mathsf{prefix}}$ is updated according to the following argument.

For a given query (K, X) to F, if $C^{\mathrm{PRE}}(V, M) = (K, X)$ for given V, μ, M such that $(V, \mu) \in \mathcal{L}_{\mathsf{prefix}}$ and $\mu || M$ in S_{prefix}, then the computation of F corresponding to this query is involved in a hash computation. Therefore, if Y is the answer returned by $\mathcal{F}$, $\mathcal{S}_{1,2}$ computes $W = C^{\mathrm{POST}}(V, M, Y)$ and adds $(W, (\mu || M))$ to $\mathcal{L}_{\mathsf{prefix}}$.

Game 2. The interceptor $\mathcal{I}_{2,3,4}$ is inserted in the Game. When receiving a query $\mu = M_1 || \ldots || M_\ell$, $\mathcal{I}$ forwards it to $\mathcal{C}$ and runs the construction by calling $\mathcal{S}$ to get the values of the internal primitive. Finally, $\mathcal{I}_{2,3,4}$, represented in Figure 5, forwards the answer returned by $\mathcal{C}$.

```
Answer to a $H$ query
Input :    $\mu \in S_{\mathsf{pad}}$
Output :   $h = \mathcal{H}(\mu)$

1. compute $(M_1||\ldots||M_\ell) = \mathsf{pad}(\mu)$
2. define $V_0 = IV$
3. for $i$ from 1 to $\ell$
   (a) define $(K, X) = C^{\mathrm{PRE}}(V_{i-1}, M_i)$
   (b) call $\mathcal{S}$ to get $Y = F(K, X)$
   (c) define $V_i = C^{\mathrm{POST}}(V_{i-1}, M_i, Y)$
4. call $\mathcal{C}$ to get $h = \mathcal{H}(\mu)$
5. return $h$
```

Fig. 5. Interceptor $\mathcal{I}_{2,3,4}$ in Games 2 to 4.

Game 3. A small modification is brought to the simulator, which is represented in Figure 6. When it receives a query to F or F^{-1}, it does not call $\mathcal{F}$ if the answer is already defined in $\mathcal{L}_{\mathsf{values}}$. It simply returns the corresponding value to the caller, either $\mathcal{I}$ or $\mathcal{D}$.

Game 4. We now define some bad events that can occur on the sequence of queries and answers involving $\mathcal{S}$. These events aim at detecting a potential bias in the answer to a hash query. The definition of bad events that may give an advantage to the adversary depend on C^{PRE}. We suppose that these events can be detected efficiently and define the corresponding subroutines of the simulator.

$$\begin{aligned}
&\mathsf{Collision\_detect} : \\
&\quad (\mathcal{L}_{\mathsf{prefix}}, V) \mapsto \begin{cases} (M, M', V') \text{ s.t. } \exists (V', \mu) \in \mathcal{L}_{\mathsf{prefix}}, C^{\mathrm{PRE}}(V, M) = C^{\mathrm{PRE}}(V', M') \\ \perp, \text{ if such } (M, M', V') \text{ does not exist} \end{cases} \\
&\mathsf{Suffix\_detect} : \\
&\quad (\mathcal{L}_{\mathsf{values}}, V) \mapsto \begin{cases} (M, X, K) \text{ s.t. } \exists (X, K, Y) \in \mathcal{L}_{\mathsf{values}}, C^{\mathrm{PRE}}(V, M) = (X, K) \\ \perp, \text{ if such } (M, X, K) \text{ does not exist} \end{cases}
\end{aligned}$$

If several answers are possible, these algorithms return one of them randomly. If this happens, $\mathcal{D}$ might be able to distinguish the two oracle systems. Therefore such cases have to be covered by the definition of bad events.

To get a significant bound on their occurrence probability, bad events have to be raised when receiving a fresh answer from $\mathcal{F}$, as these values are uniformly distributed up to the knowledge of both $\mathcal{D}$ and $\mathcal{S}$. The bad events are the following:

$\mathsf{Bad}_{\mathsf{Coll}}$. This event occurs while inserting a path (V, μ) in $\mathcal{L}_{\mathsf{prefix}}$. Suppose that there is already some (V', μ') in $\mathcal{L}_{\mathsf{prefix}}$ such that $C^{\mathrm{PRE}}(V, M) = C^{\mathrm{PRE}}(V', M')$ for some M, M' such that $\mu||M$ and $\mu'||M'$ are in $S_{\mathsf{prefix}} \cup S_{\mathsf{pad}}$. Then, $F(C^{\mathrm{PRE}}(V, M))$ is involved in the computation of two different hashes with two different prefixes, which can lead to a correlation on the digests. A noticeable case of this event occurs when $V = V'$, which leads to an internal collision on the chaining variable. If $\mathcal{D}$ can insert a common suffix to μ and μ', this leads to a collision on the digests. We refer to this event as $\mathsf{Bad}_{\mathsf{Coll}}$.

$\mathsf{Bad}_{\mathsf{Suffix}}$. Another bad event can occur at the same step of the simulation. Suppose that there exists (X, K, Y) in $\mathcal{L}_{\mathsf{values}}$ and M such that $\mu||M \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}}$ and $C^{\mathrm{PRE}}(V, M) = (K, X)$. Then, V is an intermediate chaining variable involved in a hash computation, and the next iteration of the compression function has already been executed. This can also lead to local collisions, or to digest values that are completely defined before the computation of $\mathcal{C}^{\mathcal{F}}$. We refer to this event as $\mathsf{Bad}_{\mathsf{Suffix}}$.

```
Initialization of $\mathcal{S}_{3-5}$

1. define $\mathcal{L}_{\mathsf{values}} = \emptyset$ and $\mathcal{L}_{\mathsf{prefix}} = \{(IV, \epsilon, 1)\}$
```

```
Simulation of $F$
Input :     $(K, X)$, origin $\mathcal{O} =$ either $\mathcal{I}$ or $\mathcal{D}$
Output :    $Y$

1. (Game 4 only) if $\mathcal{O} = \mathcal{D}$ and $\mathsf{Prefix\_identify}(\mathcal{L}_{\mathsf{prefix}}, K, X) = (V, \mu, 0, M)$,
   detect($\mathsf{Bad}_{\mathsf{Extension}}$)
2. if $\exists (X, K, Y, T) \in \mathcal{L}_{\mathsf{values}}$
   (a) if $T = 0$ and $\mathcal{O} = \mathcal{D}$, replace $(X, K, Y, 0)$ by $(X, K, Y, 1)$
       in $\mathcal{L}_{\mathsf{values}}$
   (b) return $Y$
3. ($\nexists (X, K, Y, T) \in \mathcal{L}_{\mathsf{values}}$) call $\mathcal{F}$ to get $Y = F(K, X)$
4. if $\mathcal{O} = \mathcal{I}$, add $(X, K, Y, 0)$ to $\mathcal{L}_{\mathsf{values}}$
5. else (if $\mathcal{O} = \mathcal{D}$), add $(X, K, Y, 1)$ to $\mathcal{L}_{\mathsf{values}}$
6. if $\mathsf{Prefix\_identify}(\mathcal{L}_{\mathsf{prefix}}, K, X) \neq \perp$
   (a) set $(V, \mu, T, M) = \mathsf{Prefix\_identify}(\mathcal{L}_{\mathsf{prefix}}, K, X)$
   (b) set $W = C^{\mathrm{POST}}(V, M, Y)$
   (c) (Games 4-5 only) if $\mathsf{Collision\_detect}(\mathcal{L}_{\mathsf{prefix}}, W)$,
       detect($\mathsf{Bad}_{\mathsf{Coll}}$)
   (d) (Games 4-5 only) if $\mathsf{Suffix\_detect}(\mathcal{L}_{\mathsf{values}}, W)$,
       detect($\mathsf{Bad}_{\mathsf{Suffix}}$)
   (e) if $\mu || M \in S_{\mathsf{prefix}}$
         i. if $T = 1$, $\mathcal{O} = \mathcal{D}$ and $(W, \mu || M, 0) \in \mathcal{L}_{\mathsf{prefix}}$, erase it
        ii. if $T = 1$ and $\mathcal{O} = \mathcal{D}$, add $(W, \mu || M, 1)$ to $\mathcal{L}_{\mathsf{prefix}}$
       iii. else add $(W, \mu || M, 0)$ to $\mathcal{L}_{\mathsf{prefix}}$
7. return $Y$
```

```
Simulation of $F^{-1}$
Input :     $(K, Y)$
Output :    $X$

1. if $\exists (X, K, Y, T) \in \mathcal{L}_{\mathsf{values}}$
   (a) if $T = 0$
         i. (Game 4 only) detect($\mathsf{Bad}_{\mathsf{Reduction}}$)
        ii. replace $(X, K, Y, 0)$ by $(X, K, Y, 1)$ in $\mathcal{L}_{\mathsf{values}}$
   (b) return $X$
2. else ($\nexists (X, K, Y) \in \mathcal{L}_{\mathsf{values}}$)
   (a) call $\mathcal{F}^{-1}$ to get $X = F^{-1}(K, Y)$
   (b) add $(X, K, Y, 1)$ to $\mathcal{L}_{\mathsf{values}}$
   (c) (Games 4-5 only) if $\mathsf{Prefix\_identify}(\mathcal{L}_{\mathsf{prefix}}, K, X) \neq \perp$,
       detect($\mathsf{Bad}_{\mathsf{Mitm}}$)
   (d) return $X$
```

Fig. 6. Simulator $\mathcal{S}_{3-5}$ for F and F^{-1} in Games 3 to 5. The instructions labelled "Game 4 only" are pointless in Game 5 and removed from the simulator.

$\mathsf{Bad}_{\mathsf{Mitm}}$. Finally, an unwanted property might occur due to the answer of queries to F^{-1}. Suppose that the oracle $\mathcal{F}$ returns X to a query (K, Y) to F^{-1} and that for some V, μ, M such that $(V, \mu) \in \mathcal{L}_{\mathsf{prefix}}$ and $\mu||M \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}}$, $C^{\mathrm{PRE}}(V, M) = (K, X)$. This might also lead to internal collisions, or hashes that are defined without running $\mathcal{C}$ on the answers of $\mathcal{F}$. We refer to this event as $\mathsf{Bad}_{\mathsf{Mitm}}$.

We can now prove the following two lemmas. Lemma 2 ensures that if no bad event occurs, $\mathcal{L}_{\mathsf{prefix}}$ contains all the chaining variables that can be identified as an intermediate computation of a digest from the values of F stored in $\mathcal{L}_{\mathsf{values}}$. Lemma 3 ensures that if no bad event occurs, the last call to F in the computation of a digest can always be identified as such for a unique message when the query is received by $\mathcal{S}$.

Lemma 2. *Let $\mathcal{L}_{\mathsf{values}}$ and $\mathcal{L}_{\mathsf{prefix}}$ denote the internal memory of $\mathcal{S}$ after answering any of the queries, and suppose that no bad event has occured during the game. Let $V_0 = IV, \ldots, V_t$ be a sequence of chaining variables, $M_1, \ldots, M_t$ be a sequence of message blocks and $(X_1, K_1, Y_1), \ldots, (X_t, K_t, Y_t)$ a sequence of elements of $\mathcal{L}_{\mathsf{values}}$ such that $\mu = M_1|| \ldots ||M_t \in S_{\mathsf{prefix}}$ and for all $1 \leq i \leq t$,*

$$C^{\mathrm{PRE}}(V_{i-1}, M_i) = (X_i, K_i)$$
$$C^{\mathrm{POST}}(V_{i-1}, M_i, Y_i) = V_i \ .$$

Then, (V_t, μ) is in $\mathcal{L}_{\mathsf{prefix}}$.

Proof. We prove this result by recurrence on t. (IV, ϵ) is added during the initialization phase of $\mathcal{S}$, which prooves the result for $t = 0$.

Now, let us suppose that it is true for all sequences of length $(t-1)$. Suppose that (X_t, K_t, Y_t) is the last of the values (X_i, K_i, Y_i) to be added to $\mathcal{L}_{\mathsf{values}}$. Then, when it is added, if no bad event has occured before, $\mathcal{L}_{\mathsf{prefix}}$ contains $(V_{t-1}, M_1|| \ldots |M_{t-1})$.

- If (X_t, K_t, Y_t) is added as the result of a query to F (as opposed to F^{-1}), $\mathcal{S}$ then adds (V_t, μ) to $\mathcal{L}_{\mathsf{prefix}}$.
- Else, X_t is obtained as $F^{-1}(K_t, Y_t)$ and $\mathsf{Bad}_{\mathsf{Mitm}}$ occurs.

We now consider the case in which (X_t, K_t, Y_t) is not the last input/output value of F added to $\mathcal{L}_{\mathsf{values}}$. We then have $t \geq 2$. If no bad event has occured, our hypothesis ensures that $(X_1, K_1, Y_1), \ldots, (X_{t-1}, K_{t-1},$ have been added in that order. The addition of $(X_{t-1}, K_{t-1}, Y_{t-1})$ then leads to a bad event, either $\mathsf{Bad}_{\mathsf{Suffix}}$ or $\mathsf{Bad}_{\mathsf{Coll}}$. This case then always leads to a bad event, which ends the proof.

Lemma 3. *Let (K, X) be the q-th query to F received by $\mathcal{S}_{3-5}$ in Game 4, and suppose that none of the previously defined bad events occur during the game. Then, one of the following propositions is true:*

- *Before the q-th query, there is a unique (V, μ, M) such that $(V, \mu) \in \mathcal{L}_{\mathsf{prefix}}$, $\mu||M \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}}$ and $C^{\mathrm{PRE}}(V, M) = (K, X)$, and it remains unique during the whole game.*
- *No such (V, μ, M) exists at the end of the game.*

Proof. We prove Lemma 3 by contraposition: if both propositions of the lemma are false, then a bad event was raised. Let (K, X) be the q-th query to F received by $\mathcal{S}_{3-5}$ in Game 4, and Y the response to this query.

Assume that there exists at the end of the game two triples $(V_0, \mu_0, M_0) \neq (V_1, \mu_1, M_1)$ such that for $i \in \{0, 1\}$, $(V_i, \mu_i) \in \mathcal{L}_{\mathsf{prefix}}$, $\mu_i||M_i \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}}$, and $C^{\mathrm{PRE}}(V_i, M_i) = (K, X)$. Without loss of generality, (V_1, μ_1) was inserted in $\mathcal{L}_{\mathsf{prefix}}$ after (V_0, μ_0). As a consequence, $\mathsf{Bad}_{\mathsf{Coll}}$ is raised when (V_1, μ_1) is inserted in $\mathcal{L}_{\mathsf{prefix}}$.

Assume now that a unique such triple (V, μ, M) exists at the end of the game, but that it did not exist before the q-th query. We thus have $C^{\mathrm{PRE}}(V, M) = (K, X)$, $\mu||M \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}}$ and (V, μ) is not in $\mathcal{L}_{\mathsf{prefix}}$ before the q-th query, but belongs to this set at the end of the game. Let us consider (K', X'), the q'-th ($q' > q$) query to $\mathcal{S}_{3-5}$ during which (V, μ) is inserted in $\mathcal{L}_{\mathsf{prefix}}$, and Y' the response to this request. Since (V, μ) is inserted in $\mathcal{L}_{\mathsf{prefix}}$, there exists (V', μ', M') such that (V', μ') is in $\mathcal{L}_{\mathsf{prefix}}$ before the q'-th query, $\mu = \mu' \| M'$ and $V = C^{\mathrm{POST}}(V', M', Y')$. Consequently, $\mathsf{Bad}_{\mathsf{Suffix}}$ is raised during query q', since (X, K, Y) has been inserted in $\mathcal{L}_{\mathsf{values}}$ during query $q < q'$ and $C^{\mathrm{PRE}}(V, M) = (K, X)$.

In Game 4, we also prepare the delay of the queries issued by $\mathcal{I}$ that will be introduced in Game 5. The three bad events that have already been defined also apply to Game 5, and their occurrence probability in Game 5 affects the final indifferentiability bound. The problem is that these probabilities are difficult to bound in Game 5, as the current knowledge of $\mathcal{D}$ is not included in the internal state of $\mathcal{S}_{3-5}$: $\mathcal{S}_{3-5}$ does not know which hash queries have been issued before the end of the game. Therefore, we define more bad events in Game 4, so that we can express an indifferentiability bound as a function of the probability that a bad event occurs in Game 4.

To be able to detect these problems, $\mathcal{S}_{3-5}$ needs to be able to determine which parts of its internal state $\mathcal{D}$ can know and which parts he cannot know. Therefore, the list $\mathcal{L}_{\mathsf{prefix}}$ is modified: its elements (V, μ) get an extra tag $T \in \{0, 1\}$ that equals 1 if and only if all the queries leading to (V, μ) have been issued by $\mathcal{D}$. When inserting a new value $(W, \mu || M, T)$ in the set $\mathcal{L}_{\mathsf{prefix}}$, $T = 1$ if and only if the tag associated to the previous chaining variable (V, μ) is 1 and the current query comes from $\mathcal{D}$. Under the same conditions, if the value $(W, \mu || M, 0)$ is already in $\mathcal{L}_{\mathsf{prefix}}$, we modify the tag value and set it to 1. It can be easily shown that when no bad event occurs, $(V, \mu, 1)$ is in $\mathcal{L}_{\mathsf{prefix}}$ if and only if all the corresponding queries have been issued by $\mathcal{D}$ as queries to F. Likewise, an extra tag $T \in \{0, 1\}$ is appended to $(X, K, Y) \in \mathcal{L}_{\mathsf{values}}$, that equals 1 if and only if (X, K, Y) corresponds to a query originating from $\mathcal{D}$.

We can now define the two other bad events.

$\mathsf{Bad}_{\mathsf{Extension}}$. This event occurs when $\mathcal{D}$ issues a query (K, X) to F such that for an element $(V, \mu, 0) \in \mathcal{L}_{\mathsf{prefix}}$ and a message block M that fulfills $\mu || M \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}}$, $C^{\mathrm{PRE}}(V, M) = (K, X)$. This event cannot be detected if $\mathcal{S}$ does not have access to hash queries.

$\mathsf{Bad}_{\mathsf{Reduction}}$. A similar event can occur when $\mathcal{D}$ issues a query (K, Y) to F^{-1}. Suppose that there is an element $(X, K, Y, 0)$ in $\mathcal{L}_{\mathsf{values}}$, $(V, \mu) \in \mathcal{L}_{\mathsf{prefix}}$ and M such that $\mu || M \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}}$ and $C^{\mathrm{PRE}}(V, M) = (K, X)$. This event can only be detected if $\mathcal{S}$ has access to hash queries.

Game 5. In this game we modify the interceptor, so that $\mathcal{I}_{5,6}$ recomputes digests only at the end of the game. The simulator is unchanged, but the bad events $\mathsf{Bad}_{\mathsf{Extension}}$ and $\mathsf{Bad}_{\mathsf{Reduction}}$ cannot occur anymore.

A relation between the probability of bad events in Games 4 and 5 can be established.

Lemma 4. *For a given instance of $\mathcal{F}$ and a given randomness of $\mathcal{D}$, the occurrences of a bad event in Games 4 and 5 are deterministic and if a bad event occurs in Game 5, a bad event also occurs in Game 4. Therefore,*

$$\Pr[\mathtt{Bad}[5]] \leq \Pr[\mathtt{Bad}[4]] \ ,$$

where the probability are defined on the randomness of $\mathcal{D}$ and a uniform selection of $\mathcal{F}$.

Proof. The first statement of the lemma comes from the fact that given a fixed $\mathcal{F}$ and a randomness of $\mathcal{D}$, the execution of Game 4 and Game 5 is deterministic.

In the following, let us assume that $\mathcal{F}$ is fixed and that the randomness of $\mathcal{D}$ is fixed. In order to prove the second statement, one has to consider every bad event type that can occur in Game 5 and show that a bad event would also occur in Game 4.

Let us first notice that the same query from $\mathcal{D}$ in Game 4 or Game 5 gets the same answer in both games: $\mathcal{C}^{\mathcal{F}}(\mu)$ if μ is a query to H, $\mathcal{F}(K, X)$ if (K, X) is a query to F, and $\mathcal{F}^{-1}(K, Y)$ if (K, Y) is a query to F^{-1}. It is then clear that $\mathcal{D}$ performs the same queries when interacting with Game 4 or with Game 5, because the queries of $\mathcal{D}$ only depend on its randomness and on the answers to its previous queries.

The difference between Game 4 and Game 5 is the behaviour of the interceptor, which directly submits to the simulator the F-queries corresponding to the execution of the construction on the H-queries in Game 4, but which delays these submissions to the end of the game in Game 5. As a consequence, $\mathcal{L}_{\mathsf{prefix}}$ at the end of Game 5 is included in $\mathcal{L}_{\mathsf{prefix}}$ at the end of Game 4. Indeed let us consider (V, μ) in $\mathcal{L}_{\mathsf{prefix}}$ at the end of Game 5.

- if $(V, \mu) = (IV, \epsilon)$, it is also in $\mathcal{L}_{\mathsf{prefix}}$ in Game 4;
- if (V, μ) was inserted in $\mathcal{L}_{\mathsf{prefix}}$ by a query originating from $\mathcal{I}$ (resp. $\mathcal{D}$), it is also in $\mathcal{L}_{\mathsf{prefix}}$ in Game 4 because the orders of queries stemming from $\mathcal{I}$ (resp. $\mathcal{D}$) is maintained in both games and the execution of queries from $\mathcal{I}$ (resp. $\mathcal{D}$) is sufficient to insert (V, μ) in $\mathcal{L}_{\mathsf{prefix}}$.

Let us assume that $\mathsf{Bad}_{\mathsf{Coll}}$ happens in Game 5, during the insertion of (V, μ) in $\mathcal{L}_{\mathsf{prefix}}$. At the time of this insertion, there exists $(V', \mu') \in \mathcal{L}_{\mathsf{prefix}}, M, M'$ such that $C^{\mathrm{PRE}}(V, M) = C^{\mathrm{PRE}}(V', M')$, and $\mu \| M$ and $\mu' \| M'$ both belong to $S_{\mathsf{prefix}} \cup S_{\mathsf{pad}}$. Thus (V, μ) and (V', μ') are also in $\mathcal{L}_{\mathsf{prefix}}$ at the end of Game 4, and $\mathsf{Bad}_{\mathsf{Coll}}$ is raised when the second one is inserted in $\mathcal{L}_{\mathsf{prefix}}$ in Game 4.

Let us now assume that $\mathsf{Bad}_{\mathsf{Suffix}}$ happens in Game 5, during the insertion of (V, μ) in $\mathcal{L}_{\mathsf{prefix}}$. At the time of this insertion, there exists $(X, K, Y) \in \mathcal{L}_{\mathsf{values}}$ and M such that $\mu \| M \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}}$ and $C^{\mathrm{PRE}}(V, M) = (K, X)$. (V, μ) is also in $\mathcal{L}_{\mathsf{prefix}}$ at the end of Game 4. If $(X, K, Y) \in \mathcal{L}_{\mathsf{values}}$ at the time of its insertion during Game 4, then $\mathsf{Bad}_{\mathsf{Suffix}}$ is raised in Game 4. Otherwise, necessarily (V, μ) is inserted by a query originating only from $\mathcal{I}$ and (X, K, Y) is inserted by a query originating from $\mathcal{D}$. At the time it is inserted in Game 4, $(V, \mu) \in \mathcal{L}_{\mathsf{prefix}}$. If the query is a F-query, $\mathsf{Bad}_{\mathsf{Extension}}$ is raised. If it is a F^{-1} query, $\mathsf{Bad}_{\mathsf{Mitm}}$ is raised.

Finally, let us assume that $\mathsf{Bad}_{\mathsf{Mitm}}$ happens in Game 5, during the insertion of (X, K, Y) in $\mathcal{L}_{\mathsf{values}}$ by a F^{-1}-query. At the time of this insertion, there exists $(V, \mu) \in \mathcal{L}_{\mathsf{prefix}}, M$ such that $\mu \| M \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}}$ and $C^{\mathrm{PRE}}(V, M) = (K, X)$. (V, μ) is also inserted in $\mathcal{L}_{\mathsf{prefix}}$ in Game 4. If it is inserted after (X, K, Y) is inserted in $\mathcal{L}_{\mathsf{values}}$, $\mathsf{Bad}_{\mathsf{Suffix}}$ happens in Game 4. If it is inserted before (X, K, Y), one has to distinguish two subcases : if (X, K, Y) is inserted first in Game 4 by a query to F, then this query was made by $\mathcal{I}$, and when the query (K, Y) to F^{-1} is made, $\mathsf{Bad}_{\mathsf{Reduction}}$ happens; otherwise $\mathsf{Bad}_{\mathsf{Mitm}}$ when (X, K, Y) is inserted in $\mathcal{L}_{\mathsf{values}}$ by a query to $\mathcal{F}^{-1}$.

Game 6. We can now withdraw the $\mathcal{F}$ oracle from the game and insert the random oracle $\mathcal{H}$. The simulator must now generate answers to F queries itself. To thwart distinguishing attacks, these answers must be consistent with the digests generated by $\mathcal{H}$, and the distance between their distribution (taken over the random coins of $\mathcal{S}$ and $\mathcal{H}$) and the one that is induced by a uniform choice of $\mathcal{F}$ must be as small as possible. To achieve it, we use the sampling algorithm Samp defined in Section 3.2.

For each query to $F(K, X)$, when there exists (V, μ) in $\mathcal{L}_{\mathsf{prefix}}$ and M such that $\mu || M \in S_{\mathsf{pad}}$ and $C^{\mathrm{PRE}}(V, M) = (K, X)$, the answer Y must be generated so that the first ℓ_h bits of $C^{\mathrm{POST}}(V, M, Y)$ equal $\mathcal{H}(\mu || M)$. Therefore $\mathcal{S}_6$ computes it by calling $\mathcal{H}(\mu || M)$ to get h and $\mathsf{Samp}(V, M, h, \mathcal{L}_{\mathsf{values}})$ to get Y.

For each query to $F(K, X)$ that does not fulfill this condition (resp. each query to $F^{-1}(K, Y)$), $\mathcal{S}_6$ selects its answer randomly and uniformly over the set of values that do not have a preimage (resp. an image) under the same key in $\mathcal{L}_{\mathsf{values}}$. We denote this set $C(\mathcal{L}_{\mathsf{values}}, K)$ (resp. $P(\mathcal{L}_{\mathsf{values}}, K)$).

A consequence of Lemma 3 is when no bad event occurs, for each H-request issued by $\mathcal{D}$, the answer returned by $\mathcal{I}$ is consistent with the sequence of values defined by $\mathcal{S}$. Therefore, as long as no bad event occurs, the only difference between Games 5 and 6 is the distribution of the values sampled by $\mathcal{S}$. As a consequence,

$$\left|\Pr\left[W_6 \wedge \overline{\mathtt{Bad}[6]}\right] - \Pr\left[W_5 \wedge \overline{\mathtt{Bad}[5]}\right]\right| \leq N_{\mathcal{D}} \Delta \ ,$$

```
Initialization of $\mathcal{I}_{5,6}$

1. define $\mathcal{L}_{\mathcal{H}} = \emptyset$
2. define $V_0 = IV$
```

```
Answer to a $H$ query
Input :    $\mu \in S_{\mathsf{pad}}$
Output :   $h = \mathcal{H}(\mu)$

1. add $\mu$ to $\mathcal{L}_{\mathcal{H}}$
2. call $\mathcal{C}$ (Game 5) or $\mathcal{H}$ (Game 6) to get $h = \mathcal{H}(\mu)$
3. return $h$
```

```
Replay of $H$ queries (at the end of the Game)

1. for each $\mu$ in $\mathcal{L}_{\mathcal{H}}$
   (a) set $(M_1 || \dots || M_\ell) = \mu$
   (b) for $i$ from $1$ to $\ell$
        i. compute $(K, X) = C^{\mathrm{PRE}}(V_{i-1}, M_i)$
       ii. call $\mathcal{S}$ to get $Y = F(K, X)$
      iii. compute $V_i = C^{\mathrm{POST}}(V_{i-1}, M_i, Y)$
```

Fig. 7. Interceptor $\mathcal{I}_{5,6}$ in Games 5 and 6.

where W_i denotes the probability that $\mathcal{D}$ returns 1 in Game i.

Furthermore, the probabilities that a bad event occurs in Games 5 and 6 fulfill

$$|\Pr[\mathsf{Bad}[6]] - \Pr[\mathsf{Bad}[5]]| \leq N_{\mathcal{D}} \varDelta \ .$$

Game 7. In Game 7, we remove $\mathcal{I}$ and the bad event detection from the game. We obtain the final simulator. In this scenario $\mathcal{D}$ interacts with $\varSigma' = (\mathcal{H}, \mathcal{S}^{\mathcal{H}})$. This does not change the view of the distinguisher

Bound on the advantage of the distinguisher. The only pair of consecutive games for which the view of $\mathcal{D}$ can be modified is Games $5-6$, when the $\mathcal{F}$-oracle is replaced by the $\mathcal{H}$-oracle. Using the difference lemma, we have:

$$\begin{aligned}
\mathsf{Adv}(\mathcal{D}) &\leq |\Pr[W_6] - \Pr[W_5]| \\
&\leq \max(\Pr[\mathsf{Bad}[5]], \Pr[\mathsf{Bad}[6]]) + N_{\mathcal{D}} \varDelta \\
&\leq \Pr[\mathsf{Bad}[5]] + 2 N_{\mathcal{D}} \varDelta \\
&\leq \Pr[\mathsf{Bad}[4]] + 2 N_{\mathcal{D}} \varDelta \ .
\end{aligned}$$

The probability of a bad event in Game 4 and the value of $\varDelta$ are investigated in the next section.

B.2 Occurrence probability of bad events for concrete hash functions

In this section, we study the occurrence probability of a bad event in Game 4. For each of the defined bad events, we study its occurrence probability when answering to each of the $N_{\mathcal{D}}$ queries, assuming no bad event has occurred before.

```
Initialization of $\mathcal{S}_{6-7}$

1. define $\mathcal{L}_{\mathsf{values}} = \emptyset$ and $\mathcal{L}_{\mathsf{prefix}} = \{(IV, \epsilon, 1)\}$
```

```
Simulation of $F$
Input :     $(K, X)$
Output :    $Y$

1. if $\exists(X, K, Y) \in \mathcal{L}_{\mathsf{values}}$
   (a) return $Y$
2. if $\mathsf{Prefix\_identify}(\mathcal{L}_{\mathsf{prefix}}, K, X) = (V, \mu, M)$ and $\mu||M \in S_{\mathsf{pad}}$
   (a) call $\mathcal{H}$ to get $h = \mathcal{H}(\mu||M)$
   (b) run $\mathsf{Samp}(M, V, h, \mathcal{L}_{\mathsf{values}})$ to get $Y$
3. else
   (a) select $Y$ uniformly over $C(\mathcal{L}_{\mathsf{values}}, K)$
4. add $(X, K, Y)$ to $\mathcal{L}_{\mathsf{values}}$
5. if $\mathsf{Prefix\_identify}(\mathcal{L}_{\mathsf{prefix}}, K, X) = (V, \mu, M)$
   (a) set $W = C^{\mathrm{POST}}(V, M, Y)$
   (b) (Game 6 only) if $\mathsf{Collision\_detect}(\mathcal{L}_{\mathsf{prefix}}, W)$,
       detect($\mathsf{Bad}_{\mathsf{Coll}}$)
   (c) (Game 6 only) if $\mathsf{Suffix\_detect}(\mathcal{L}_{\mathsf{values}}, W)$,
       detect($\mathsf{Bad}_{\mathsf{Suffix}}$)
   (d) if $\mu||M \in S_{\mathsf{prefix}}$, add $(W, \mu||M)$ to $\mathcal{L}_{\mathsf{prefix}}$
6. return $Y$
```

```
Simulation of $F^{-1}$
Input :     $(K, Y)$
Output :    $X$

1. if $\exists(X, K, Y) \in \mathcal{L}_{\mathsf{values}}$
   (a) return $X$
2. else ($\nexists(X, K, Y) \in \mathcal{L}_{\mathsf{values}}$)
   (a) select $X$ uniformly over $P(\mathcal{L}_{\mathsf{values}}, K)$
   (b) add $(X, K, Y)$ to $\mathcal{L}_{\mathsf{values}}$
   (c) (Game 6 only) if $\mathsf{Prefix\_identify}(\mathcal{L}_{\mathsf{prefix}}, K, Y) \neq \perp$,
       detect($\mathsf{Bad}_{\mathsf{Mitm}}$)
   (d) return $X$
```

Fig. 8. Simulator $\mathcal{S}_{6-7}$ for F and F^{-1} in Games 6 and 7

Bad event caused by an answer to a query to F. First, let us suppose that the q-th query is a query to $F(K_q, X_q)$. If Y_q is not already defined, $\mathcal{S}$ queries $\mathcal{F}$. The answer is uniformly distributed over a set of at least $2^n - q$ values. By definition of λ_{Coll}, for each chaining variable V such that there is a (V, μ) in $\mathcal{L}_{\mathsf{prefix}}$, the number of answers that generate $\mathsf{Bad}_{\mathsf{Coll}}$ is at most λ_{Coll}.

Similarly, for each (X, K, Y) in $\mathcal{L}_{\mathsf{values}}$, the number of answers that lead to $\mathsf{Bad}_{\mathsf{Suffix}}$ is bounded by a constant $\lambda_{\mathsf{Suffix}}$ that happens to be equal to $\lambda_{\mathsf{Extension},1}$. For any functions C^{POST} and C^{PRE}, it can be easily shown that $\lambda_{\mathsf{Suffix}} \leq \lambda_{\mathsf{Coll}}$.

Each of the previous $q-1$ queries leads to the definition of a triple (X, K, Y) and belongs to one of the following cases.

1. There is no $(V, \mu), M$ with $(V, \mu) \in \mathcal{L}_{\mathsf{prefix}}$ and $\mu || M \in S_{\mathsf{prefix}} \cup S_{\mathsf{pad}}$ such that $C^{\mathrm{PRE}}(V, M) = (K, Y)$. Then, the value (K, X) has to be taken into account when evaluating $\mathsf{Bad}_{\mathsf{Suffix}}$.
2. There is one such $(V, \mu), M$. Then, for the q-th query, if (K, X) leads to $\mathsf{Bad}_{\mathsf{Suffix}}$, (V, μ) leads to $\mathsf{Bad}_{\mathsf{Coll}}$. (K, X) does not need to be taken into account.

For the q-th query, we can conclude that

$$\Pr\left[\mathsf{Bad}_{\mathsf{Coll}}[4](q) \vee \mathsf{Bad}_{\mathsf{Suffix}}[4](q)\right] \leq \frac{q\lambda_{\mathsf{Coll}}}{2^n - q} .$$

Bad event caused by an answer to a query to F^{-1}. Now, we suppose that the q-th query is a query to $F^{-1}(K_q, Y_q)$. If X_q is not already known to $\mathcal{S}$, $\mathcal{S}$ queries $\mathcal{F}$ and gets an answer which is uniformly distributed in a set of at least $2^n - q$ values. By definition of λ_{Mitm}, for each chaining variable V such that there is a $(V, \mu) \in \mathcal{L}_{\mathsf{prefix}}$, the number of answers that generate $\mathsf{Bad}_{\mathsf{Mitm}}$ is λ_{Mitm}. Therefore,

$$\Pr\left[\mathsf{Bad}_{\mathsf{Mitm}}[4](q)\right] \leq \frac{q\lambda_{\mathsf{Mitm}}}{2^n - q} .$$

Merging the probabilities of a bad event on the answers of $\mathcal{F}$. As $\mathsf{Bad}_{\mathsf{Mitm}}$ cannot occur on the same type of queries as $\mathsf{Bad}_{\mathsf{Coll}}$ and $\mathsf{Bad}_{\mathsf{Suffix}}$, we have

$$\Pr\left[\mathsf{Bad}_{\mathsf{Coll}}[4](q) \vee \mathsf{Bad}_{\mathsf{Suffix}}[4](q) \vee \mathsf{Bad}_{\mathsf{Mitm}}[4](q)\right] \leq \frac{q\max(\lambda_{\mathsf{Coll}}, \lambda_{\mathsf{Mitm}})}{2^n - q} .$$

By summing on all the queries, we get that

$$\Pr\left[\mathsf{Bad}_{\mathsf{Coll}}[4] \vee \mathsf{Bad}_{\mathsf{Suffix}}[4] \vee \mathsf{Bad}_{\mathsf{Mitm}}[4]\right] \leq \frac{\max(\lambda_{\mathsf{Coll}}, \lambda_{\mathsf{Mitm}})N_{\mathcal{D}}^2}{2(2^n - N_{\mathcal{D}})} .$$

$\mathsf{Bad}_{\mathsf{Extension}}$. Now, we study the occurrence probability of the $\mathsf{Bad}_{\mathsf{Extension}}$ event. This event occurs if $\mathcal{D}$ guesses a value (K, X) for which there is a value $(W, \mu || M, 0)$ in $\mathcal{L}_{\mathsf{prefix}}$ and a message block M' such that $C^{\mathrm{PRE}}(W, M') = (K, X)$. For each such $(W_i, \mu_i || M_i, 0)$, $\mathcal{D}$ can choose M'_i and knows at most $h_i = W_i|_{\ell_h}$ (by a direct query to H), M_i, and V_i such that $(V_i, \mu_i, 1) \in \mathcal{L}_{\mathsf{prefix}}$ and $C^{\mathrm{POST}}(V_i, M_i, Y_i) = W_i$. First, we deal with the case of h_i that the adversary knows.

By definition of λ_{Pre}, each h_i is in one of the sets S_j of maximal size λ_{Pre} that form a partition of $\{0,1\}^{\ell_h}$. For each guess (K, X), W_i can lead to $\mathsf{Bad}_{\mathsf{Extension}}$ only if (K, X) can be reached from the set S_j h_i belongs to. It can be reached from only one such set, denoted $S(K, X)$. We denote by Ω_t the event that there is at least t such values h_i that belong to the same set S_j.

For a guess (K, X) and a value $(W_i, \mu_i || M_i)$ such that $W_i|_{\ell_h}$ belongs to the set $S(K, X)$, we now estimate the probability that (K, X) is the right value using the knowledge of the adversary. This probability is taken over a random distribution of Y_i. By definition of μ_{Post}, there are at least $\mu_{\mathsf{Post}} - q$ values Y_i that lead to h_i (if $\mu_{\mathsf{Post}} - q < 1$, there is at least one such value). By definition of $\lambda_{\mathsf{Extension},2}$, at most $\lambda_{\mathsf{Extension},2}$ of these values lead to (K, X). Therefore we get that

$$\Pr\left[\mathsf{Bad}_{\mathsf{Extension}}[4](q) | \Omega_t \wedge \overline{\Omega}_{t+1}\right] \leq t\frac{\lambda_{\mathsf{Extension},2}}{\mu_{\mathsf{Post}} - q}$$

By summing over all the queries, we get

$$\Pr\left[\mathsf{Bad}_{\mathsf{Extension}}[4] | \Omega_t \wedge \overline{\Omega}_{t+1}\right] \leq t \frac{\lambda_{\mathsf{Extension},2} N_{\mathcal{D}}}{\mu_{\mathsf{Post}} - N_{\mathcal{D}}}$$

Now, as Ω_{t+1} implies Ω_t we sum over t and get

$$\begin{aligned}\Pr\left[\mathsf{Bad}_{\mathsf{Extension}}[4]\right] &\leq \frac{\lambda_{\mathsf{Extension},2} N_{\mathcal{D}}}{\mu_{\mathsf{Post}} - N_{\mathcal{D}}} \sum_{t=1}^{N_{\mathcal{D}}} (t(\Pr\left[\Omega_t\right] - \Pr\left[\Omega_{t+1}\right])) \\ &\leq \frac{\lambda_{\mathsf{Extension},2} N_{\mathcal{D}}}{\mu_{\mathsf{Post}} - N_{\mathcal{D}}} \sum_{t=1}^{N_{\mathcal{D}}} \Pr\left[\Omega_t\right] \ .\end{aligned}$$

The distribution of each h_i implies a distribution of the set S_j it belongs to. The event Ω_t can then be interpreted as the existence of a t-collision over at most $N_{\mathcal{D}}$ variables, where each value is chosen with probability at most $\frac{\lambda_{\mathsf{Pre}}}{2^{\ell_h} - N_{\mathcal{D}}}$.

To give a more explicit bound, we use the following result.

Lemma 5. *Let Ω_t be the event that there exists a subset of at least t identical values in a set of N values $\{x_1, \ldots, x_N\}$, drawn independently and randomly following a distribution D such that for all y, $\Pr\left[x_i = y\right] \leq p$. Then,*

$$\sum_{t=1}^{N} \Pr\left[\Omega_t\right] \leq \min\left(N, (1+e)Np - 2\log_2 p\right) \ .$$

Proof. It is well-known that for any $t > 1$,

$$\Pr\left[\Omega_t\right] \leq \frac{N^t}{\sqrt{2\pi t}} \left(\frac{e}{t}\right)^t p^{t-1}.$$

We now pose

$$t_0 = eNp - 2\log_2 p \ ,$$

and prove that for any $t \geq t_0$ and any N,

$$\frac{N^t}{\sqrt{2\pi t}} \left(\frac{e}{t}\right)^t p^{t-1} \leq p. \tag{1}$$

Indeed, (1) is equivalent to

$$t\left(\log_2 t - \log_2 p - \log_2 N - \log_2 e\right) + \frac{1}{2}\log_2 t \geq -2\log_2 p - \frac{1}{2}\log_2(2\pi).$$

Then for any $t \geq t_0$,

$$\log_2 t = \log_2\left(eNp\right) + \log_2\left(1 - \frac{2\log_2 p}{eNp}\right) = \log_2 e + \log_2 N + \log_2 p + \log_2\left(1 - \frac{2\log_2 p}{eNp}\right) \ ,$$

wherefrom

$$t\left(\log_2 t - \log_2 p - \log_2 N - \log_2 e\right) = -2\log_2 p\left(1 - \frac{eNp}{2\log_2 p}\right)\log_2\left(1 - \frac{2\log_2 p}{eNp}\right) \geq -2\log_2 p$$

where the last inequality is deduced from the fact that for $x \geq 0$,

$$f : x \mapsto \left(1 + \frac{1}{x}\right)\log_2(1+x)$$

is increasing and $\lim_{x\to 0} f(x) = 1/\ln 2 > 1$. Using $t \geq -2\log_2 p \geq 1$, this implies

$$t\left(\log_2 t - \log_2 p - \log_2 N - \log_2 e\right) + \frac{1}{2}\log_2 t \geq -2\log_2 p \ .$$

Therefore we get

$$\sum_{t=1}^{N} \Pr\left[\Omega_t\right] = \sum_{t=1}^{\lceil t_0\rceil - 1} \Pr\left[\Omega_t\right] + \sum_{t=\lceil t_0\rceil}^{N} \Pr\left[\Omega_t\right] \leq \lceil t_0\rceil - 1 + Np \ ,$$

and finally

$$\sum_{t=1}^{N} \Pr\left[\Omega_t\right] \leq (1+e)Np - 2\log_2 p \ .$$

Now, we deal with the case where the adversary does not know h_i. Therefore, to his knowledge, Y_i is uniformly distributed over a set of at least $2^n - N_{\mathcal{D}}$ values, and at most $\lambda_{\mathsf{Extension},1}$ such values lead to his guess (K, X). In that case, the success probability for the q-th query is bounded by $q/(2^n - N_{\mathcal{D}})$.

By taking the maximum probability of both cases, we get

$$\Pr\left[\mathsf{Bad}_{\mathsf{Extension}}[4]\right] \leq \max\left(\frac{g(N_{\mathcal{D}}, \lambda_{\mathsf{Pre}}, \ell_h)\lambda_{\mathsf{Extension},2} N_{\mathcal{D}}}{\mu_{\mathsf{Post}} - N_{\mathcal{D}}}, \frac{\lambda_{\mathsf{Extension},1} N_{\mathcal{D}}^2}{2(2^n - N_{\mathcal{D}})}\right) \ .$$

Prefix-free encoding. In the specific case of prefix-free encodings, the demonstration above can be slightly improved. For each of the W_i, the adversary does not know h_i. Therefore, only the second case of the demonstration above applies. This leads to

$$\Pr\left[\mathsf{Bad}_{\mathsf{Extension}}[4]\right] \leq \frac{\lambda_{\mathsf{Extension},1} N_{\mathcal{D}}^2}{2(2^n - N_{\mathcal{D}})} \ .$$

$\mathsf{Bad}_{\mathsf{Reduction}}$. For the reverse attacks, the computation is similar to the case of length-extension attacks. For each h_i, the adversary has to guess a value Y_i among all those that lead to h_i after C^{POST}. By definition, there are at least $\mu_{\mathsf{Post}} - q$ such values. For each guess Y only some values h_i are compatible: those such that the actual value Y belongs to $S'(V_i, M_i, h_i)$. By definition of h_i, the probability that they $S'(V_i, M_i, h_i)$ is a given subset S'_j of the partition depends on the distribution of Y_i and is bounded by $\frac{\lambda_{\mathsf{Post}}}{2^n - N_{\mathcal{D}}}$. Therefore, with a demonstration similar to the case of length extension attacks, we can prove that

$$\Pr\left[\mathsf{Bad}_{\mathsf{Reduction}}[4]\right] \leq \frac{g(N_{\mathcal{D}}, \lambda_{\mathsf{Post}}, n) N_{\mathcal{D}}}{\mu_{\mathsf{Post}} - N_{\mathcal{D}}} \ .$$

Bounding the probability of a guess by $\mathcal{D}$. $\mathsf{Bad}_{\mathsf{Extension}}$ and $\mathsf{Bad}_{\mathsf{Reduction}}$ can occur for different types of queries. Therefore, in the generic case,

$$\Pr\left[\mathsf{Bad}_{\mathsf{Extension}}[4] \vee \mathsf{Bad}_{\mathsf{Reduction}}[4]\right] \leq \max\left(g(N_{\mathcal{D}}, \lambda_{\mathsf{Pre}}, \ell_h)\lambda_{\mathsf{Extension},2}, g(N_{\mathcal{D}}, \lambda_{\mathsf{Post}}, n)\right) \frac{N_{\mathcal{D}}}{\mu_{\mathsf{Post}} - N_{\mathcal{D}}} \ .$$

This concludes the proof of Theorem 1.