Invited Paper

Journal of Information Processing Systems, Vol.6, No.4, December 2010 DOI : 10.3745/JIPS.2010.6.4.453

Security Properties of Domain Extenders for Cryptographic Hash Functions

Elena Andreeva*, Bart Mennink* and Bart Preneel*

Abstract—Cryptographic hash functions reduce inputs of arbitrary or very large length to a short string of fixed length. All hash function designs start from a compression function with fixed length inputs. The compression function itself is designed from scratch, or derived from a block cipher or a permutation. The most common procedure to extend the domain of a compression function in order to obtain a hash function is a simple linear iteration; however, some variants use multiple iterations or a tree structure that allows for parallelism. This paper presents a survey of 17 extenders in the literature. It considers the natural question whether these preserve the security properties of the compression function, and more in particular collision resistance, second preimage resistance, preimage resistance and the pseudo-random oracle property.

Keywords—Hash Functions, Domain Extenders, Security Properties

1. INTRODUCTION

A hash function $H:\{0,1\}^* \rightarrow \{0,1\}^n$ is a function that maps arbitrarily long bit strings (or at least very long strings) to digests of fixed length. They were introduced in cryptology in the seminal paper of Diffie and Hellman on public-key cryptography [1] with as main goal to make digital signatures more efficient and compact: the idea is that one would sign a hash value of a message rather than a message itself. In his 1979 PhD thesis [2], Merkle stated the three main security properties of a hash function: collision resistance, second preimage resistance and preimage resistance. Cryptographic hash functions can be used in a broad range of applications: to compute a short unique identifier of an input string (for a digital signature as mentioned above), as one-way functions to hide an input string (e.g. for passphrase protection), to commit to a string in a protocol, for entropy extraction and for key derivation. Hash functions became a very popular tool during the 1990s because MD5 is 10 times faster than DES in software; moreover, it offered a larger security level than DES, could deal with short and long inputs, and posed less problems under export control laws. As a consequence, hash functions were even used to construct MAC algorithms, stream ciphers and block ciphers.

In order to accommodate the processing of messages of arbitrary size, similarly to encryption

※ This work has been funded in part by the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy), in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II, and in part by the Research Council K.U.Leuven: GOA TENSE. The first author is supported by a Ph.D. Fellowship from the Flemish Research Foundation (FWO-Vlaanderen). The second author is supported by a Ph.D. Fellowship from the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

Manuscript received November 4, 2010; accepted November 5, 2010.

Corresponding Author: Bart Preneel

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

modes of operation, most hash functions are designed by reusing small and fixed input length functions, known as *compression functions*, under some composition method. A compression function F is a hash function whose message space $\mathcal{M}$ is of a fixed size. We refer to the hash function under composition as a *domain extender*. The level of hash function modularity can be further refined by building compression functions on top of other building blocks, such as block ciphers and permutations. We will briefly elaborate on this in Sect. 7.

The most straightforward way to construct a domain extender is an *iterative* composition, that is constructed starting from a compression function $F:\{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$. One first pads the input string, divides it into exactly ℓ b-bit blocks m_i and computes iteratively $h_i \leftarrow F(h_{i-1} \| m_i)$ and returns h_ℓ. The first example of such a design is the Rabin hash function [3], with $F(x \| y) = DES_y(x)$. Early designs did not specify the initial value $IV = h_0$. It is easy to see that this leads to trivial second preimage attacks and collision attacks, even if F is second preimage resistant: it suffices to remove the first input block and select $IV = h_1$ for the second message. Similarly, finding preimages for the Rabin hash function is trivial if the IV can be chosen by the attacker.

A natural question is to ask which conditions should be imposed on the compression function and on the iteration mechanism for the hash function to be secure. For collision resistance, this question was answered by Damgård [4] and Merkle [5] in two independent papers published at Crypto'89. They both showed that if one has if one fixes the IV and has an unambiguous padding scheme with the message length appended at the end, it is sufficient that F is collision resistant for H to be collision resistant. Lai and Massey called this construction Merkle-Damgård strengthening [6]; the strengthened iteration described here is now commonly referred to as the Merkle-Damgård design. Subsequently Lai and Massey claimed that a hash function H is ideally second preimage resistant, that is, it takes about 2^n steps to find a second preimage for F, if and only if F is ideally second preimage resistant; unfortunately, this result turns out to be incorrect. Obtaining ideal (second) preimage resistance seems to be difficult.

Until 2005, MD5 and SHA-1 were the most widely used hash functions; this is surprising, as collisions for the compression function of MD5 were already published in 1993 by den Boer and Bosselaers [7] and in 1996 by Dobbertin [8]. While these attacks were serious warnings, no one managed to find collisions for MD5 itself. However, in 2004 Wang et al. [9, 10] achieved a breakthrough on both MD5 and SHA-1: they made several clever improvements to differential cryptanalysis in order to find collisions for MD5 in 15 minutes on a PC and to speed up collision search for SHA-1 with a factor 2000. Around the same time, several results were published that showed subtle flaws in the Merkle-Damgård design [11-14]. These developments sparked a strong interest in the topic of hash functions, resulting in more cryptanalysis, new research on definitions [15], novel domain extenders and reduction proofs and new hash function constructions. As a consequence of this hash function crisis, NIST decided to launch in 2007 a new hash function competition to select by 2012 the new SHA-3 standard [16].

Six years after the beginning of the hash function crisis and three years after the start of the SHA-3 competition, this paper presents a comparative overview of the 17 domain extenders for hash functions that are currently known. In particular, we identify 8 security properties for hash functions and analyze to which extent the known domain extenders preserve these properties. Our work is relevant to the SHA-3 competition, but has also broader implications on the theory of hash functions.

The remainder of this paper is organized as follows. In Sect. 2 we briefly summarize prelimi-

naries on hash function theory. The Merkle-Damgård design and its security properties are discussed in Sect. 3. The ideas behind variations of the Merkle-Damgård design are considered in Sect. 4. Section 5 contains a list of Merkle-Damgård-based domain extenders, together with their security properties, and in Sect. 6 we elaborate on other domain extenders not directly related to the Merkle-Damgård design. In Sect. 7, we refine the level of modularity by considering hash function security properties with respect to idealization of the underlying permutations or block ciphers. Then, in Sect. 8, the findings of this work are applied to NIST's SHA-3 hash function competition. The work is concluded in Sect. 9.

2. DEFINITIONS

Notation. For $n \in \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers, let $\{0,1\}^n$ denote the set of bit strings of length n, $\{0,1\}^{n^*}$ the set of strings of length a multiple of n and $\{0,1\}^*$ the set of strings of arbitrary length. If x, y are strings, then $x \| y$ is the concatenation of x and y. If $k, l \in \mathbb{N}$ then $\langle k \rangle_l$ is the encoding of k as an l-bit string. $|x|$ denotes the bit size of the string x. If S is a set, then $x \xleftarrow{\$} S$ denotes the uniform random selection of an element from S. We let $y \leftarrow \mathrm{A}(x)$ and $y \xleftarrow{\$} \mathrm{A}(x)$ be the assignment to y of the output of a deterministic and randomized algorithm A, respectively, when run on input x.

2.1 Design of Hash Functions

Keyless and keyed hash functions. A keyless hash function is defined as $\mathrm{H}: \mathcal{M} \to \mathcal{Y}$. The message space $\mathcal{M}$ could be infinitely large, but we assume that there exists a $\lambda \in \mathbb{N}$ such that $\{0,1\}^\lambda \subseteq \mathcal{M}$, and the target space $\mathcal{Y}$ is a finite set of bit strings. A keyed hash function takes an additional key input parameter from the finite key space $\mathcal{K}$ and is formally defined as $\mathrm{H}: \mathcal{K} \times \mathcal{M} \to \mathcal{Y}$. The key is a public parameter that indexes the concrete hash instance of the hash function family H. We note that many existing hash functions, including SHA-1 and SHA-2, are unkeyed.

2.2 Security of Hash Functions

Security notions. In [15], Rogaway and Shrimpton investigate seven security notions for keyed hash functions as a natural extension of the three basic keyless notions of *collision resistance* (Coll), *preimage resistance* (Pre), and *second preimage resistance* (Sec). Four more notions emerge in the keyed setting and these are namely the *always*- and *everywhere*-variants of second preimage and preimage resistance (aPre, aSec, ePre, and eSec). Intuitively, collision resistance means that, for random key $K \xleftarrow{\$} \mathcal{K}$ it is hard for an adversary to find different messages M, M' such that $\mathrm{H}(K, M) = \mathrm{H}(K, M')$. Second preimage resistance means that it, given a key K and first preimage M, it is hard for an adversary to find a message M' such that $\mathrm{H}(K, M) = \mathrm{H}(K, M')$. In the original second preimage notion, both the key and message are generated at random, but the adversary may be possible to arbitrarily choose the key (always second preimage resistance) or the message (everywhere second preimage resistance). The preimage resistance notion differs from the second preimage resistance in the sense that the adversary learns a range point Y of H, rather than a preimage M. More formally, we obtain the following definitions [15].

Definition 1. *Let* $\mathrm{H}: \mathcal{K} \times \mathcal{M} \to \mathcal{Y}$ *be a hash function. The advantage of an adversary* A *in breaking a keyed hash function* H *under security notion* $\mathrm{atk} \in \{\mathrm{Coll}, \mathrm{Sec}[\lambda], \mathrm{eSec}, \mathrm{aSec}[\lambda]\}$ *is given by*

$$\mathbf{Adv}^{\mathrm{atk}}_{\mathrm{H}}(\mathrm{A}) = \Pr[\mathrm{Exp}_{\mathrm{atk}}: M \neq M' \text{ and } \mathrm{H}(K, M) = \mathrm{H}(K, M')], \tag{1}$$

and under security notion $\mathrm{atk} \in \{\mathrm{Pre}[\lambda], \mathrm{ePre}, \mathrm{aPre}[\lambda]\}$ *by*

$$\mathbf{Adv}^{\mathrm{atk}}_{\mathrm{H}}(\mathrm{A}) = \Pr[\mathrm{Exp}_{\mathrm{atk}}: \mathrm{H}(K, M') = Y], \tag{2}$$

where the experiments $\mathrm{Exp}_{\mathrm{atk}}$ *are given below.*

atk	$\mathrm{Exp}_{\mathrm{atk}}$
Coll	$K \xleftarrow{\$} \mathcal{K}\ ;\ (M, M') \xleftarrow{\$} \mathsf{A}(K)$
Sec[λ]	$K \xleftarrow{\$} \mathcal{K}\ ;\ M \xleftarrow{\$} \{0,1\}^{\lambda}\ ;\ M' \xleftarrow{\$} \mathsf{A}(K, M)$
eSec	$(M, st) \xleftarrow{\$} \mathsf{A}\ ;\ K \xleftarrow{\$} \mathcal{K}\ ;\ M' \xleftarrow{\$} \mathsf{A}(K, st)$
aSec[λ]	$(K, st) \xleftarrow{\$} \mathsf{A}\ ;\ M \xleftarrow{\$} \{0,1\}^{\lambda}\ ;\ M' \xleftarrow{\$} \mathsf{A}(M, st)$
Pre[λ]	$K \xleftarrow{\$} \mathcal{K}\ ;\ M \xleftarrow{\$} \{0,1\}^{\lambda}\ ;\ Y \leftarrow \mathrm{H}(K, M)\ ;\ M' \xleftarrow{\$} \mathsf{A}(K, Y)$
ePre	$(Y, st) \xleftarrow{\$} \mathsf{A}\ ;\ K \xleftarrow{\$} \mathcal{K}\ ;\ M' \xleftarrow{\$} \mathsf{A}(K, st)$
aPre[λ]	$(K, st) \xleftarrow{\$} \mathsf{A}\ ;\ M \xleftarrow{\$} \{0,1\}^{\lambda}\ ;\ Y \leftarrow \mathrm{H}(K, M)\ ;\ M' \xleftarrow{\$} \mathsf{A}(Y, st)$

The hash function H *is called* (t, ε) atk*-secure if no adversary running in time at most* t *has advantage more than* ε.

Remark. The notions of everywhere preimage and second preimage security become meaningless in the keyless setting because the adversary wins trivially the security game. Indeed, in the keyless setting, for everywhere (second) preimage resistance the probabilities in (1) and (2) contain no randomness. The notions of always second preimage and preimage security on the other hand, merge with the standard second and preimage notion in the keyless setting, respectively. Now, for the keyless setting, the definitions for preimage and second preimage resistance (Def. 1) read as follows: the advantage of an adversary A in breaking a keyless hash function H under security notion $\mathrm{atk} \in \{\mathrm{Sec}[\lambda], \mathrm{Pre}[\lambda]\}$ is given by $\mathbf{Adv}^{\mathrm{atk}}_{\mathrm{H}}(\mathrm{A}) = \Pr[\mathrm{Exp}_{\mathrm{atk}}: M \neq M' \text{ and } \mathrm{H}(M) = \mathrm{H}(M')]$ for $\mathrm{atk} = \mathrm{Sec}[\lambda]$, and by $\mathbf{Adv}^{\mathrm{atk}}_{\mathrm{H}}(\mathrm{A}) = \Pr[\mathrm{Exp}_{\mathrm{atk}}: \mathrm{H}(M') = Y]$ for $\mathrm{atk} = \mathrm{Pre}[\lambda]$, where the experiments $\mathrm{Exp}_{\mathrm{atk}}$ are given below.

atk	$\mathrm{Exp}_{\mathrm{atk}}$
Sec[λ]	$M \xleftarrow{\$} \{0,1\}^{\lambda}\ ;\ M' \xleftarrow{\$} \mathsf{A}(M)$
Pre[λ]	$M \xleftarrow{\$} \{0,1\}^{\lambda}\ ;\ Y \leftarrow \mathrm{H}(M)\ ;\ M' \xleftarrow{\$} \mathsf{A}(Y)$

Redefining the collision security in a similar fashion to the second preimage and preimage notions is however not formally correct in the keyless setting, as was pointed out by Damgård [4]. The problem with such a definition is that there always exist an efficient attack algorithm that outputs a collision with probability 1; namely the algorithm that has hard-coded in it one of the many collisions. This definitional problem was further investigated by Stinson [17] and Rogaway [18] who proposed a different theoretical treatment by using security reductions. In Rogaway's interpretation, one tackles the stated "human-ignorance" problem by assuming the exis-

tence of explicitly given reduction(s). Such a treatment says that for a collision secure hash function there is an explicitly given reduction of the following form: given an adversary A against a scheme using internally H, there is a corresponding, explicitly-specified adversary B, as efficient as A, for finding collisions in H. This is a restatement of a standard reduction in cryptography meant to capture the idea that if someone knows how to break the higher-level scheme then they know how to find collisions in H, and if nobody can find collisions in H then nobody can break the scheme. The notions defined above in both keyed and keyless setting are referred to as *standard* security assumptions of the cryptographic strength of a hash function.

Indifferentiability. To prove the security of practical systems or schemes, e.g. digital signatures, one frequently uses to the ideal (random oracle) security model where the hash function is assumed to be an idealized function or a random oracle. A random oracle [19, 20] is a public function which returns random outputs for each new input query. An adversary that queries inputs to the random oracle function is said to have only "black-box" access to the function.

While random oracles are monolithic objects, real world hash functions most commonly are highly structured, for example by following some composition method, as is the case with the Merkle-Damgård design. To formalize a notion that compares the behavior of such hash functions to that of a random oracle, one may decide to use the definition of PRF (pseudorandom function) based on the indistinguishability concept. However, the PRF notion does not always capture the overall behavior of such highly structured primitives. In particular, most hash function designs employ a publicly computable compression function which implies that anyone can compute the intermediate state values of the hash function and reconstruct the hash result himself. This particular strength of a real world adversary is not reflected in the notion of indistinguishability (it becomes meaningless in the absence of randomness like a secret key) and thus resulted in the improved notion of indifferentiability of a hash function from a random oracle.

The indifferentiability framework was introduced by Maurer et al. [21] as an extension of the classical notion of indistinguishability, and further developed in the context of hash functions by Coron et al. [11]. It proves that if a hash function H^{F} based on an ideal compression function subcomponent F is indifferentiable from an ideal primitive $\mathcal{R}$ (random oracle), then H^{F} can replace $\mathcal{R}$ in any system. Unlike for the security properties of Def. 1, in the indifferentiability framework we consider information-theoretic adversaries. These adversaries are computationally unbounded, and their advantages are measured in the number of queries made to the oracles.

Definition 2. *A hash function* H *with oracle access to an ideal compression function primitive* F *is said to be* $(t_D, t_S, q, \varepsilon)$ *indifferentiable from an ideal primitive* $\mathcal{R}$ *if there exists a simulator* $\mathcal{S}$*, such that for any distinguisher* D *it holds that:*

$$\mathbf{Adv}^{\mathrm{PRO}}_{\mathrm{H},\mathcal{S}}(\mathrm{D}) = \left| \Pr\left[\mathrm{D}^{\mathrm{H}^{\mathrm{F}},\mathrm{F}} = 1\right] - \Pr\left[\mathrm{D}^{\mathcal{R},\mathcal{S}^{\mathcal{R}}} = 1\right] \right| < \varepsilon. \tag{3}$$

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

The distinguisher D converses either with the real world $(\mathrm{H}^{\mathrm{F}}, \mathrm{F})$ or the simulated world

$(\mathcal{R}, \mathcal{S}^{\mathcal{R}})$, and its goal is to tell both worlds apart.

Provable security approach. In the provable security framework, one argues xxx security of the domain extender hash function H under some assumption on the yyy security of the underlying compression function F. We say that H is (t, ε) xxx-secure if any adversary A running in time at most t has ε probability of success in breaking the xxx security of H. Similarly, we define the security of the compression function F. Now, assuming a secure compression function F with respect to property yyy under the provable security design paradigm, we prove security of H for property xxx. That allows to upper bound the advantage $\mathbf{Adv}^{\text{xxx}}_{\text{H}}$. If yyy is identical with xxx, we speak of security preservation and when we have a weaker than xxx security goal for yyy we speak of property amplification. Hash function security preservation results in the standard model for $\text{yyy} \in \{\text{Coll}, \text{Sec}[\lambda], \text{eSec}, \text{aSec}[\lambda], \text{Pre}[\lambda], \text{ePre}, \text{aPre}[\lambda]\}$ are discussed broadly in [15, 22] and in Sect. 5.

3. The Merkle-Damgård Design and Security Properties

As mentioned in Sect. 1, the most adopted approach for hash function domain extenders is the iterative composition: let the compression function $\text{F}: \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$ take as inputs a chaining or state variable h of size n bits and a message block m of size b bits, and output the updated chaining variable of size n bits. In order to allow for input messages of arbitrary length, the Merkle-Damgård hash function needs an injective padding pad, that transforms M into a message $M' = \text{pad}(M)$ of length a multiple of the block size. However, as becomes clear in Sect. 4.2, a simple injective padding is not sufficient. In particular, we will consider the Merkle-Damgård design with length strengthening, due to Merkle [5], or *strengthened Merkle-Damgård* [6]. This strengthened design uses a padding function $\text{ls}-\text{pad}$ that appends the encoding of the message length at the end of the message to generate the padded message $M' = \text{ls}-\text{pad}(M)$. Then, M' is processed as a sequence of message blocks $m_1 \| \ldots \| m_\ell$ with $|m_i| = b$ bits for $i = 1, \ldots, \ell$. Additionally, the strengthened Merkle-Damgård design employs a fixed IV, for *initialization vector* (cf. Sect. 1). This value will be the first state value of the Merkle-Damgård design.

Now, we can define the Merkle-Damgård hash function $\mathcal{SMD}$ as follows:

Algorithm $\mathcal{SMD}_{\text{F}}(M)$:
  $m_1 \| \ldots \| m_\ell \leftarrow \text{ls-pad}(M)$
  $h_0 \leftarrow IV$
  For $i = 0, \ldots, \ell$ do $h_i \leftarrow \text{F}(h_{i-1} \| m_i)$
  Return h_ℓ.

The established padding function $\text{ls}-\text{pad}$ for the Merkle-Damgård extender is a specific form of a *suffix-free* padding. A suffix-free padding ensures that $X \| \text{pad}(M) \neq \text{pad}(M')$ for all $M \neq M'$ and all arbitrary bit strings X. A distinct padding rule is the *prefix-free* one where for any distinct M, M', there exists no bit string X such that $\text{pad}(M') = \text{pad}(M) \| X$. One example of a prefix-free hash function includes the message length in bits as a part of the first message block. Note that this padding rule is highly inefficient for long messages, the length of

which is not known in advance. In the remainder, by $\mathrm{sf}-\mathrm{pad}$ and $\mathrm{pf}-\mathrm{pad}$ we denote any suffix-free and prefix-free padding algorithm, respectively.

From a practical perspective [3], the iterative principle of message processing offers the benefit of computing the message digest "on the fly". More precisely, the message is processed in chunks of message blocks which are small enough to call the Merkle-Damgård design a streaming mode of operation. We clarify that classical streaming modes work on bit level, whereas here we mean blockwise streaming. Other than temporarily storing message bits up to a full block size ready to be evaluated, the Merkle-Damgård hash function stores temporarily also the intermediate chaining state result (until the next message block arrives). On the negative side, the Merkle-Damgård design does not permit for parallel processing which offers a potential speed-up if multiple processors are available.

From a security perspective, the Merkle-Damgård design with length strengthening ($\mathcal{SMD}$) offers Coll security guarantees by means of property preservation [5, 4]. The Sec and Pre preservation of the $\mathcal{SMD}$ domain extender and its variants remained long underexplored. In [22], Andreeva et al. showed that the iteration preserves neither of the two properties.[1] A result by Dean [23] and Kelsey and Schneier [14] shows that the Merkle-Damgård iteration looses a factor linear in the message length (in blocks) of the second preimage security when the underlying compression function is assumed to be ideal.

If the underlying compression function of the $\mathcal{SMD}$ extender is a keyed function, then $\mathcal{SMD}$ preserves only the ePre security notion and fails in the preservation of aPre, aSec and eSec [22].

The indifferentiability analysis shows that the Merkle-Damgård domain extender does not behave like a random oracle. A concrete counterexample in the framework of Maurer et al. [21] was exhibited in the work of Coron et al. [11]. A much earlier attack, known as the *length extension* attack, also exemplifies the non-random behavior of the Merkle-Damgård extender. Let $h_1 = \mathrm{H}(M_1)$ be the hash result of the message M_1 where M_1 is unknown to the adversary, but the length $|M_1|$ is known. It is easy for the adversary to append a suffix M_2 and compute the hash value $h_2 = \mathrm{H}(M_1 \| M_2)$. Such an attack should be infeasible in $\min\{2^{|M_1|}, |\mathcal{Y}|\}$ hash function evaluations for an ideal function as a domain extender and shows that the Merkle-Damgård construction deviates from an ideal function.

4. MERKLE-DAMGÅRD VARIANTS: CLASSIFICATION AND PROPERTIES

Most popular domain extenders nowadays use the Merkle-Damgård construction internally. In our attempt to provide a summary of the main Merkle-Damgård-based domain extenders, we first classify them according to several criteria.

4.1 Classification

Wide- versus narrow-pipe domain extenders. Analyzing the recent designs submitted to the NIST SHA-3 hash function competition [16], we distinguish two main design strategies: *narrow-pipe* and *wide-pipe*. The original wide-pipe design was introduced by Lucks [24], and is

1 We note that, albeit [22] considers $\mathcal{SMD}$ in the keyed setting, the results carry over to the setting where $\mathcal{SMD}$ is keyless: the proof of preservation of Coll and the non-preservation of Sec and Pre does not explicitly use the fact that the design is keyed.

characterized by keeping a full large internal state in the iterative Merkle-Damgård portion. As final step, a distinct final output transformation is employed on this "wide" state to compress it to the desired output hash length, which is shorter than the internal state size. The concept of wide-pipe designs is generalized as a generic transform that encompasses hash functions which process a large internal state (as a result of internal compression function calls) and produce their outputs by invoking a distinct output transformation function. Note that the final transformation could also evaluate additional inputs other than the state value, such as message, fixed padding, or counter bits. Several second round SHA-3 candidates have adopted the wide-pipe strategy, namely Blue Midnight Wish [25], CubeHash [26], ECHO [27], Fugue [28], Grøstl [29], JH [30], Keccak [31], Luffa [32], SIMD [33] and Shabal [34].

Narrow-pipe constructions, introduced by Rabin [3], are designated by iterating a state as large as the output hash value. These constructions may also contain an optional output transformation or other features but in essence are iterative designs with a *narrow state*. Second round SHA-3 narrow-pipe designs are BLAKE [35], Hamsi [36], SHAvite-3 [37] and Skein [38].

Keyed versus keyless domain extenders. Another separation of domain extenders is based on the presence or lack of an explicit key input. When the key is unique for every message, we refer to it as *salt*. Keyed designs are often less efficient than keyless but come with more security guarantees. Many designs that have advanced in the NIST competition include them as an optional input.

To clarify the use of keys in domain extenders in relation to the preservation security results of Andreeva et al. [22], we provide a short discussion on the interpretation of these results. All of the domain extenders in the work of Andreeva et al. are analyzed in the keyed setting. Namely, the authors consider that the underlying compression function(s) are keyed and in that sense the whole domain extender is rendered keyed. Thus, the security results for all investigated domain extenders deal with property-preservation of the seven main security notions of Rogaway and Shrimpton [15] (see Sect. 2.2). In our work, we deal with seven security property preservation only when the underlying compression function is explicitly keyed by design. Otherwise (when the compression function is keyless) for both keyed and keyless domain extenders, we only consider the three properties Coll, $\mathrm{Sec}[\lambda]$, and $\mathrm{Pre}[\lambda]$ (see Sect. 2.2).

We clarify that all findings of Andreeva et al. [22] translate easily to the ones presented here. This is done by just ignoring the key input to the compression function in the work of Andreeva et al. that is possible because the relevant security results here are independent of whether or not the compression function is keyed. Thus, all results of analyzed keyed constructions in Andreeva et al. are reducible to the seven or three main (overlapping) notions that are relevant for the treatment of keyed versus keyless domain extenders in this work.

4.2 Security Properties: High Level Intuition

In this section we provide a general discussion on Merkle-Damgård variants with respect to their (1) security guarantees in the standard model achieved by means of security preservation making a standard security assumption on the compression function(s), and (2) indifferentiability results in the random oracle model when the compression function(s) is viewed as an ideal function.

Collision security. The $\mathcal{SMD}$ design preserves Coll security due to its suffix-free padding. Suffix-free padding as a means of collision preservation was further generalized by Andreeva, Mennink and Preneel [39] to facilitate the analysis of the second round SHA-3 hash function candidates. The theorem applies to any suffix-free Merkle-Damgård-based construction that allows for an additional collision secure output transformation and/or a possible chopping at the end. For convenience, this theorem is included in App.A. On the negative side, Merkle-Damgård-based designs without suffix-free padding, need not result in collision resistance preservation as shown by Bellare and Ristenpart [40].

Second preimage and preimage security. Second preimage resistance of the Merkle-Damgård domain extender was first studied in the work of Lai and Massey [6]. In [22], it was shown that most of the Merkle-Damgård variants do not preserve Sec and Pre. The counterexamples that exhibit the lack of preservation exploit the possible loss of entropy in the iteration of the state value. Most often loss of entropy is due to the introduction of fixed bits through the state input by the initialization vector and possibly through the message input. For example, typical counterexamples for Sec and Pre security consist of constructing an underlying compression function that outputs IV if the state input equals IV, but acts like a Sec / Pre secure compression function for all other inputs. Another security weakness that additionally hurdles the preservation are the fixed padding message bits. Such non-random inputs to the compression function lead to insecure Merkle-Damgård style domain extenders, unless some message and state randomization is applied. The approach of randomizing the compression function inputs in the iterative portion of the domain extender was for instance taken by [22, 41].

Always and everywhere second preimage and preimage security. The security notions aPre, aSec, ePre, and eSec are only valid for keyed domain extenders. The general preservation recipe for the second preimage and preimage security is applicable for aSec and aPre. To achieve eSec, the randomization of the fixed padding bits is not required since an eSec adversary controls the message bits and therefore the constant padding bits. Surprisingly, the property of ePre is the easiest to satisfy and is preserved by all keyed domain extenders. The reason for that is straightforward: the preservation of ePre depends on the ePre security of the final compression function which is ePre by the original assumption.

Pseudorandom oracle behaviour. The length extension attack is a clearly demonstrate that the domain extender is differentiable from a random oracle. An obvious way to avoid extension attacks is to apply an independent output transformation at the end of a Merkle-Damgård iterative hash function. Now, under the assumption that the internal iterated compression function F, as well as the independent output transformation G are ideal functions, Coron et al. [11] show that the resulting composition is a PRO. They also observed that iterative constructions that chop the final bits of the output, or with present prefix-free padding also succumb to the extension attacks and allow for indifferentiability proofs. For a series of other indifferentiability results on hash functions, we refer to [42-44].

Another structural approach to indifferentiability was exhibited by Dodis, Ristenpart and Shrimpton [45]. Instead of assuming an ideal compression function F in the iteration, they relax the idealness assumption by substituting it with the notion of *preimage awareness*. Preimage awareness is a weaker notion than idealness (but a stronger notion than collision resis-

tance), and intuitively it means that if an adversary outputs a range value of H for later use, then he must already know a preimage of it. Preimage awareness is a notion which is preserved under iterative composition. Additionally, when a distinct final ideal output transformation G is applied, the resulting domain extender also exhibits ideal behavior in the sense of indifferentiability.

If a hash function design is proven indifferentiable, it means that the function behaves like a random oracle. In particular, it guarantees that, up to a certain degree, the design is secured against any generic attack, such as finding preimages, collisions, multicollisions, etc., in the ideal model where the adversary has query access to the ideal compression function [39, Thm. 1].

While these results are very helpful for generic compositions, often the underlying compression function exhibits a highly non-ideal behavior, which makes most of the above result inapplicable. In such cases, the indifferentiability proof is attempted by posing assumptions on the idealness of the smaller scale components, e.g. permutations, block ciphers, etc. (see Sect. 7).

5. Concrete Merkle-Damgård Alternatives and Their Security Properties

Next, we summarize the main characteristics and security properties of some of the most prominent theoretical constructions in the hash function literature, as opposed to new proposals to the NIST competition. This is a conscious decision aiming to provide an insight into the design characteristics and security properties of theoretical domain extenders, which give inspiration for the practical ones. A similar survey for the second round SHA-3 candidates is conducted in the work of Andreeva, Mennink and Preneel [39].

The algorithms of the theoretical domain extenders are given in Fig. 1. In Table 1, the security

Table 1. A summary of the security results of the theoretical domain extenders discussed in this work. The symbol "✓" indicates that the notion is provably preserved by the domain extender, and "✗" means that it is not preserved. The symbol "?" means that no result is known, and "—" is used to indicate that the security notion is irrelevant for the (keyless) domain extender. The security notions are explained in Sect. 2.2, and the domain extenders in Sects. 3, 5 and 6.

Scheme	Coll	Sec	aSec	eSec	Pre	aPre	ePre	PRO
SMD [5, 4]	✓ [5, 4]	✗ [22]	—	—	✗ [22]	—	—	✗ [11]
PfMD [11]	✗ [40]	✗ [22]	—	—	✗ [22]	—	—	✓ [11]
EMD [40]	✓ [40]	✗ [22]	—	—	✗ [22]	—	—	✓ [40]
MDP [47]	✓ [39]	✗	—	—	✗	—	—	✓ [47]
LH [48]	✗ [22]	✗ [22]	✗ [22]	✗ [48]	✗ [22]	✗ [22]	✓ [22]	✓ [40]
XLH [48]	✓ [22]	✗ [22]	✗ [22]	✓ [48]	✗ [22]	✗ [22]	✓ [22]	✗ [49]
SH [50]	✓ [22]	✗ [22]	✗ [22]	✓ [50]	✗ [22]	✗ [22]	✓ [22]	✗ [49]
ROX [22]	✓ [22]	✓ [22]	✓ [22]	✓ [22]	✓ [22]	✓ [22]	✓ [22]	✗ [51]
BCM [41]	✓ [41]	✓ [41]	—	—	✗ [41]	—	—	?
HAIFA [52]	✓ [5, 4]	✗ [22]	—	—	✗ [22]	—	—	✓ [11]
Dither [54]	✓ [5, 4]	✗ [22]	—	—	✗ [22]	—	—	?
RMX [55]	✓ [22]	✗ [22]	—	—	✗ [22]	—	—	✗ [51]
SMT [57]	✓ [22]	✗ [22]	—	—	✗ [22]	—	—	?
TH [48]	✗ [22]	✗ [22]	✗ [22]	✗ [22]	✗ [22]	✗ [22]	✓ [22]	?
XTH [48]	?	?	✗ [22]	?	✓ [22]	✗ [22]	✓ [22]	?
LDP [24]	✓ [39]	✗ [22]	—	—	✗ [22]	—	—	✓ [64]
Zipper [65]	?	✗ [22]	—	—	✗ [22]	—	—	✓ [65]

Algorithm $\mathcal{SMD}_{\mathrm{F}}(M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ ls-pad(M) ; $h_0 \leftarrow IV$ For $i = 1 \ldots \ell$ do $h_i \leftarrow \mathrm{F}(h_{i-1} \| m_i)$ Return h_ℓ	Algorithm $\mathcal{PFMD}_{\mathrm{F}}(M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ pf-pad(M) ; $h_0 \leftarrow IV$ For $i = 1 \ldots \ell$ do $h_i \leftarrow \mathrm{F}(h_{i-1} \| m_i)$ Return h_ℓ
Algorithm $\mathcal{EMD}_{\mathrm{F}}(M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ emd-pad(M) ; $h_0 \leftarrow IV$ For $i = 1 \ldots \ell - 1$ do $h_i \leftarrow \mathrm{F}(h_{i-1} \| m_i)$ Return $h_\ell \leftarrow \mathrm{F}(IV' \| m_\ell \| h_{\ell-1})$	Algorithm $\mathcal{MDP}_{\mathrm{F}}(M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ ls-pad(M) ; $h_0 \leftarrow IV$ For $i = 1 \ldots \ell - 1$ do $h_i \leftarrow \mathrm{F}(h_{i-1} \| m_i)$ Return $h_\ell \leftarrow \mathrm{F}(\pi(h_{\ell-1}) \| m_\ell)$
Algorithm $\mathcal{LH}_{\mathrm{F}}(K_1 \| \ldots \| K_\ell, M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ ls-pad(M) ; $h_0 \leftarrow IV$ For $i = 1 \ldots \ell$ do $h_i \leftarrow \mathrm{F}(K_i, h_{i-1} \| m_i)$ Return h_ℓ	Algorithm $\mathcal{XLH}_{\mathrm{F}}(K \| K_1 \| \ldots \| K_\ell, M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ ls-pad(M) ; $h_0 \leftarrow IV$ For $i = 1 \ldots \ell$ do $h_i \leftarrow \mathrm{F}(K, (h_{i-1} \oplus K_{i-1}) \| m_i)$ Return h_ℓ
Algorithm $\mathcal{SH}_{\mathrm{F}}(K \| K_1 \| \ldots \| K_{\lceil \log \ell \rceil}, M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ ls-pad(M) ; $h_0 \leftarrow IV$ For $i = 1 \ldots \ell$ do $h_i \leftarrow \mathrm{F}(K, (h_{i-1} \oplus K_{\nu(i)}) \| m_i)$ Return h_ℓ	Algorithm $\mathcal{ROX}_{\mathrm{F}}(K, M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ rox-pad(M) ; $h_0 \leftarrow IV$ For $i = 1 \ldots \lfloor \log(\ell) \rfloor$ do $\mu_i \leftarrow \mathrm{G}_1(K, M\|_k^{\mathrm{lsb}}, \langle i \rangle)$ For $i = 1 \ldots \ell$ do $h_i \leftarrow \mathrm{F}(K, (h_{i-1} \oplus \mu_{\nu(i)}) \| m_i)$ Return h_ℓ
Algorithm $\mathcal{BCM}_{\mathrm{F}}(K_1 \| K_2 \| K_3, M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ ls-pad(M) ; $h_0 \leftarrow IV \oplus K_1$ For $i = 1 \ldots \ell - 2$ do $h_i \leftarrow \mathrm{F}((m_{i+1}\|_n^{\mathrm{msb}} \oplus h_{i-1}) \| m_i)$ $h_{\ell-1} \leftarrow \mathrm{F}(((m_\ell \oplus K_2)\|_n^{\mathrm{msb}} \oplus h_{\ell-2}) \|$ $(m_{\ell-1} \oplus (0^{b-n} \| K_1)))$ Return $h_\ell \leftarrow \mathrm{F}((h_{\ell-1} \oplus K_3) \| (m_\ell \oplus K_2))$	Algorithm $\mathcal{HAIFA}_{\mathrm{F}}(M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ hai-pad(M) ; $h_0 \leftarrow IV$ $S \xleftarrow{\$} \{0,1\}^s$ // S *is a salt* For $i = 1 \ldots \lceil \|M\|/b \rceil$ do $h_i \leftarrow \mathrm{F}(h_{i-1} \| m_i \| \langle \min\{bi, \|M\|\} \rangle_t \| S)$ If $\ell > \|M\|/b\rceil$ do $h_\ell \leftarrow \mathrm{F}(h_{\ell-1} \| m_\ell \| \langle 0 \rangle_t \| S)$ Return S, h_ℓ
Algorithm $\mathit{Dither}_{\mathrm{F}}(M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ ls-pad(M) ; $h_0 \leftarrow IV$ $z_1 \| \ldots \| z_\ell \in \{0,1\}^{16\ell}$ a dithering sequence For $i = 1 \ldots \ell$ do $h_i \leftarrow \mathrm{F}(h_{i-1} \| m_i \| z_i)$ Return h_ℓ	Algorithm $\mathcal{RMX}_{\mathrm{F}}(R, M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ sf-pad(M) $h_0 \leftarrow \mathrm{F}(R \| IV)$ For $i = 1 \ldots \ell$ do $h_i \leftarrow \mathrm{F}(h_{i-1} \| (m_i \oplus R))$ Return h_ℓ
Algorithm $\mathcal{SMT}_{\mathrm{F}}(M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ t-pad(M) For $j = 1 \ldots a^d$ do $h_{0,j} \leftarrow m_j$ For $i = 1 \ldots d$ and $j = 1 \ldots a^{d-i}$ do $h_{i,j} \leftarrow \mathrm{F}(h_{i-1,(j-1)a+1} \| \cdots \| h_{i-1,ja})$ $h_{d+1,1} \leftarrow \mathrm{F}(h_{d,1} \| \langle \|M\| \rangle_{n(a-1)})$ Return $h_{d+1,1}$	Algorithm $\mathcal{XTH}_{\mathrm{F}}(K \| K_1 \| \ldots \| K_{d+1}, M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ t-pad(M) For $j = 1 \ldots a^d$ do $h_{0,j} \leftarrow m_j$ For $i = 1 \ldots d$ and $j = 1 \ldots a^{d-i}$ do $h_{i,j} \leftarrow \mathrm{F}(K, (h_{i-1,(j-1)a+1} \| \cdots \| h_{i-1,ja}) \oplus K_i)$ $h_{d+1,1} \leftarrow \mathrm{F}(K, (h_{d,1} \| \langle \|M\| \rangle_{n(a-1)}) \oplus K_{d+1})$ Return $h_{d+1,1}$
Algorithm $\mathcal{TH}_{\mathrm{F}}(K_1 \| \ldots \| K_{d+1}, M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ t-pad(M) For $j = 1 \ldots a^d$ do $h_{0,j} \leftarrow m_j$ For $i = 1 \ldots d$ and $j = 1 \ldots a^{d-i}$ do $h_{i,j} \leftarrow \mathrm{F}(K_i, h_{i-1,(j-1)a+1} \| \cdots \| h_{i-1,ja})$ $h_{d+1,1} \leftarrow \mathrm{F}(K_{d+1}, h_{d,1} \| \langle \|M\| \rangle_{n(a-1)})$ Return $h_{d+1,1}$	Algorithm $\mathcal{LDP}_{\mathrm{F}}(M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ ls-pad(M) ; $h_0, h'_0 \leftarrow IV, IV'$ For $i = 1 \ldots \ell - 1$ do $h_i \leftarrow \mathrm{F}(h_{i-1} \| h'_{i-1} \| m_i)$ $h'_i \leftarrow \mathrm{F}(h'_{i-1} \| h_{i-1} \| m_i)$ Return $h_\ell \leftarrow \mathrm{F}(h_{\ell-1} \| h'_{\ell-1} \| m_\ell)$
Algorithm $\mathit{Zipper}_{\mathrm{F}}(M)$: $m_1 \| \ldots \| m_\ell \leftarrow$ sf-pad(M) ; $h_0 \leftarrow IV$ For $i = 1 \ldots \ell$ do $h_i \leftarrow \mathrm{F}(h_{i-1} \| m_i)$ For $i = 1 \ldots \ell$ do $h_{\ell+1} \leftarrow \mathrm{F}(h_{\ell+i-1} \| m_{\ell-i+1})$ Return $h_{2\ell}$	Padding algorithms: ls-pad$(M) = M \| 10^{-\|M\|-1-t \bmod b} \| \langle \|M\| \rangle_t$ emd-pad$(M) = M \| 10^{-\|M\|-1-64-n \bmod b} \| \langle \|M\| \rangle_{64}$ hai-pad$(M) = M \| 10^{-\|M\|-1-t-r \bmod b} \| \langle \|M\| \rangle_t \| \langle \|n\| \rangle_r$ t-pad$(M) = M \| 1 \| 0^{na\lceil \log_a((\|M\|+1)/n) \rceil - \|M\|-1}$ rox-pad$(M) =$ first $\lceil (\|M\| + 2n)/b \rceil \cdot b$ bits of $M \| \mathrm{G}_2(M\|_k^{\mathrm{lsb}}, \langle \|M\| \rangle, \langle 1 \rangle) \| \mathrm{G}_2(M\|_k^{\mathrm{lsb}}, \langle \|M\| \rangle, \langle 2 \rangle) \| \ldots$

Fig. 1. Iterations $\mathcal{SMD}, \mathcal{PFMD}, \mathcal{EMD}, \mathcal{RMX}, \mathcal{MDP}, \mathcal{BCM}$ and use compression function $\mathrm{F}: \{0,1\}^{n+b} \to \{0,1\}^n$; $\mathcal{LH}, \mathcal{XLH}, \mathcal{SH}$ and $\mathcal{ROX}$ use $\mathrm{F}: \{0,1\}^k \times \{0,1\}^{n+b} \to \{0,1\}^n$; $\mathcal{LDP}$ uses $\mathrm{F}: \{0,1\}^{2n+b} \to \{0,1\}^n$; $\mathcal{SMT}$ uses $\mathrm{F}: \{0,1\}^{an} \to \{0,1\}^n$; $\mathcal{TH}$ and $\mathcal{XTH}$ use $\mathrm{F}: \{0,1\}^k \times \{0,1\}^{an} \to \{0,1\}^n$; $\mathcal{HAIFA}$ and Dither use $\mathrm{F}: \{0,1\}^{n+b+l+s} \to \{0,1\}^n$, where for Dither, $l = 16$ and $s = 0$. Iteration $\mathcal{EMD}$ requires the parameters to satisfy $b \geq n + 64$, $\mathcal{BCM}$ requires $b \geq n$, and $\mathcal{HAIFA}$ requires $b \geq r + t$. The function $\nu(i)$ is the largest integer j such that $2^j \mid i$. G_1 and G_2 are described in Sect. 5. $\pi: \{0,1\}^n \to \{0,1\}^n$ is a random permutation.

properties of the constructions are summarized. Recall that the security of the $\mathcal{SMD}$ construction is already analyzed in Sect. 4.2. We note that this work does not consider the tightness of the security reductions (cf. [15]). These formal bounds can be found in the corresponding references.

Prefix-free Merkle-Damgård. The basic prefix-free Merkle-Damgård [11] ($\mathcal{PFMD}$) designs are narrow-pipe, keyless iterative domain extenders that apply a prefix-free padding function pf−pad (cf. Sect. 2).

Prefix-free domain extenders are proven to be indifferentiable from a random oracle in the work of Coron et al. [11]. If the prefix-free designs are not as well suffix-free, they do not preserve Coll , but they do if the padding is also suffix-free [22]. Irrespectively of the presence of suffix-free padding, the prefix-free constructions also fail to achieve preservation of Sec and Pre [22].

Enveloped Merkle-Damgård. The enveloped Merkle-Damgård [40] ($\mathcal{EMD}$) was proposed by Bellare and Ristenpart and resembles the design of HMAC [46]. It is a narrow-pipe, keyless domain extender. $\mathcal{EMD}$ uses two fixed initialization vectors IV and IV'. The first vector is applied in a Merkle-Damgård style as input to the first compression function. The second IV' is provided as input to the final compression function together with the chaining variable and the final input message bits and this step is known as the "enveloping" step of the construction.

$\mathcal{EMD}$ is proven to be Coll preserving and indifferentiable from a random oracle in [40]. The suffix-free padding ensures the collision preservation. The "enveloping" is applied to hide the internal Merkle-Damgård construction and guarantees the indifferentiability result. Similar enveloping domain extenders are previously used by the NMAC and HMAC constructions [46] to build PRFs out of compression functions, both proven to be indifferentiable transforms in [11]. In [22], the $\mathcal{EMD}$ hash function is shown not to be Sec and Pre preserving.

Merkle-Damgård with permutation. The Merkle-Damgård with permutation, due to Hirose, Park and Yun [47], ($\mathcal{MDP}$) domain extender is a narrow-pipe, keyless variant of the original Merkle-Damgård design. The difference with the Merkle-Damgård construction is that a permutation is applied before the processing of the last message block.

The permutation masks the internal Merkle-Damgård style processing, similarly to the idea of $\mathcal{EMD}$, and $\mathcal{MDP}$ is proven indifferentiable from a random oracle when the underlying compression function is an ideal function [47]. $\mathcal{MDP}$ also preserves Coll security due to the suffix-free Merkle-Damgård padding [39], but does not achieve Sec and Pre security preservation by analogy to the Merkle-Damgård style designs analyzed in Andreeva et al. [22]. As a way of example, these non-preservation results are proven in App.B.

Linear hash. The linear hash function as described by Bellare and Rogaway [48] ($\mathcal{LH}$) is a narrow-pipe, keyed Merkle-Damgård domain extender. The only difference with the Merkle-Damgård design is that it accepts an additional key input in every call of the iteration. Moreover, each key is distinct and therefore $\mathcal{LH}$ requires number of key inputs that is a linear in the message size. Notice that this approach ensures a domain separation of the underlying compression function, and another way to view the construction is to assume that it employs distinct compression functions for each message block evaluation.

Bellare and Rogaway showed in [48] that the design preserves eSec in case of equal-length messages (thus in case $\mathcal{LH}$ is a fixed input length domain extender), but that it does not preserve eSec in the general case (where the target and forged message may be of different lengths). Additionally, the extender preserves ePre resistance, but it fails to conserve the remaining four properties [22]. Bellare and Ristenpart [40] prove that domain extenders that apply a distinct final compression function are indifferentiable from a RO. This result applies to $\mathcal{LH}$.

Linear XOR. The linear XOR by Bellare and Rogaway [48] ($\mathcal{XLH}$) is a narrow-pipe, keyed Merkle-Damgård domain extender. In contrast to the $\mathcal{LH}$ hash function it adds the same number of distinct keys by XORing these with the chaining values resulting from each iteration of the Merkle-Damgård style hash function. The first key is XORed with the initialization vector IV and the final key is XORed with the final intermediate chaining value, while the final hash result is left unmodified.

The $\mathcal{XLH}$ domain extender is shown to preserve eSec in [48]. Additionally, [22] showed that this construction preserves Coll and ePre, and does not preserve the remaining security properties. $\mathcal{XLH}$ is not is differentiable from a RO, by virtue of an attack described by [49].

Shoup's hash. Shoup's hash function [50] ($\mathcal{SH}$) derives from the linear XOR hash function and optimizes it in terms of the number of keys. It uses logarithmic number of keys (instead of linear), following a specific sequence; the sequence was proven in [50] to be the optimal to suffice to prove everywhere second preimage security (similarly to the latter two designs).

The security results are identical to the results for $\mathcal{XLH}$.

ROX. The ROX [22] ($\mathcal{ROX}$) domain extender is a narrow-pipe, keyed hash function. It draws largely from the XOR-linear hash and Shoup's hash. $\mathcal{ROX}$ requires a logarithmic number of masks, instead of keys (as in the Shoup's hash function), to be XORed with the chaining values. The masks are generated by applying a function G_1 to a sequence of strings $(K, \mu, \langle i \rangle_n)$ (for $i = 1, \ldots, \lfloor \log(\ell) \rfloor$) that consist of the compression function key K of length k, the first k bits of the message, and an encoding of a counter i. A function G_2 (optionally G_2 could be identical with G_1) is applied on inputs the first message bits μ, the length encoding of the processed message length λ and a counter i of the necessary invocations of the function to create the padding string. The padding function is suffix- but not prefix-free.

From a security perspective, the suffix-free padding ensures the Coll security preservation, and ePre security is trivially preserved. Everywhere second preimage security is argued similarly to the latter two designs. The novelty here is the preservation of the remaining four security properties. This is due to the randomization provided both by the masks in the iteration and by the padding scheme chosen; however, the result is partially in the random oracle model where G_1 and G_2 are assumed to be random functions and the iterated compression function F is realized in the standard model and achieves standard model security guarantees. $\mathcal{ROX}$ is not PRO as a result of an attack described in [51].

BCM. The backwards chaining mode by Andreeva and Preneel [41] ($\mathcal{BCM}$) is a narrow-pipe, keyed hash function. It uses three keys K_1, K_2 and K_3 of fixed length $(b+2n)$ bits, where $|K_2| = b$ and $|K_1| = |K_3| = n$ where n is the state and b is the block size. It XORs the key K_1 and the most significant n bits of block m_2 with the fixed initial chaining variable IV. The message block m_1 together with the resulting value from the XOR computation form the input to the first application of F. In the iteration the message block m_i and the chaining variable h_{i-1} in-line are XORed with the most significant n bits of the next-in-line message block m_{i+1} and form the inputs to the i-th compression function F. The one but last block $m_{\ell-1}$ is interpreted differently than the rest of the message blocks. Here the difference is that the least significant n bits of $m_{\ell-1}$ are XORed with the key K_1, while the chaining variable $h_{\ell-2}$ is XORed with the first significant bits of K_2 and m_ℓ. The final input to the last com-

pression function is provided by the last message block m_ℓ and the chaining variable $h_{\ell-1}$ XORed with keys K_2 and K_3, respectively.

The idea behind the $\mathcal{BCM}$ domain extender is the preservation of the main security properties in the standard model eliminating the need of random functions to generate masking input as is the case with the $\mathcal{ROX}$ construction. $\mathcal{BCM}$ succeeds in attaining Coll and Sec preservation and the latter is possible due to the randomization of the internal chaining values with message blocks down the line and the external states and the padding bits with the provided three keys. $\mathcal{BCM}$ additionally preserves Pre security in the random oracle model assuming the underlying compression function is an ideal function. No indifferentiability result is known for the $\mathcal{BCM}$ construction.

HAIFA. The HAIFA design by Biham and Dunkelman [52] ($\mathcal{HAIFA}$) is a narrow-pipe hash function characterized by the inclusion of an bit counter of size usually 64 bits (to accommodate long messages) and an optional key input in every invocation of the compression function. HAIFA is often referred to as a framework or a general transformation technique that can be applied to any domain extender that is defined with the latter two characteristics. The idea is incorporated also in three second round SHA-3 candidates: BLAKE [35], ECHO [27] and SHAvite-3 [37].

The inclusion of a bit counter ensures the suffix and prefix properties of the design and helps to prove it Coll secure and indifferentiable from a random oracle [39, 11]. HAIFA does not, however, achieve Sec and Pre security in the preservation sense [22]. An alternative result in the random oracle model shows that HAIFA achieves optimal (2^n) second preimage security if the underlying compression function is assumed to be a random oracle [53].

Dither hash. The dither hash function by Rivest [54] ($\mathcal{D}ither$) is a narrow-pipe, keyless hash function. Similarly to the HAIFA hash function it includes an additional counter-like input. And while HAIFA introduces a bit counter and requires its size to accommodate long messages, the design intension behind the dither construction is to decrease the number of bits used for this extra input to either 2 or 16 bits. This increases the bandwidth available for actual data. The additional input, called the "dithering" input, to the compression function is formed by the consecutive elements of a fixed sequence. In his proposal Rivest suggested the use the infinite abelian square-free sequence [54].

With respect to Coll, Sec and Pre preservation, the same results as for $\mathcal{SMD}$ apply to the $\mathcal{D}ither$ construction. No indifferentiability result is known for the $\mathcal{D}ither$ hash function.

Randomized hash. Randomized Hashing or RMX by Halevi and Krawczyk [55] ($\mathcal{RMX}$), similarly to the HAIFA framework, can be adopted as a transform for any domain extension method. The $\mathcal{RMX}$ transform is in its essence a message modification technique. It prepends a random string R to the message as a first message block to be processed and then the same random string is XORed with each message block. The idea is to randomize the message inputs by XOR-ing a key (salt) input into the message. $\mathcal{RMX}$ was proposed as a general transform that is particularly well-suited for digital signature applications of hash functions. It aims the provision of security guarantees even when the compression function is compromised with respect to collision security. It was formally showed that just finding collisions on the compression function is not sufficient in order to break the resultant signatures: instead, the attacker needs to solve a

much harder cryptanalytical problem, closer to finding second preimages.

The randomized hash mode of operation was hence originally proved to be everywhere second preimage secure by making stronger (than everywhere second preimage) assumptions on the underlying compression function. In the security analysis of $\mathcal{RMX}$ [22] treating the value R as either randomness per message or long term key yields identical results with respect to seven property preservation. There, it is proven that $\mathcal{RMX}$ preserves Coll and ePre, but none of the other notions are preserved. No indifferentiable results are known for the $\mathcal{RMX}$ transform. In [56], Gauravaram and Knudsen demonstrated an existential forgery attack for the $\mathcal{RMX}$-hash-then-sign signature scheme, where the $\mathcal{RMX}$ hash function employs the Davies-Meyer compression function. However, this result does not contradict the security claims of [55].

6. Other Domain Extenders

In the quest for Merkle-Damgård alternatives, apart from the ones described in Sect. 5, a number of domain extenders have appeared in literature that do not directly extend the original Merkle-Damgård design. In this section, we discuss some *tree-based hash functions* and *multi-pass hash functions*. The incentives of these designs are twofold: increasing the efficiency rate and/or the security guarantees. The theoretical domain extenders discussed in this section are also included in Fig. 1 and Table 1.

6.1 Tree-Based Hash Functions

The *tree-based* constructions, in contrast to the Merkle-Damgård based designs, allow for parallelism. While Merkle-Damgård-based designs require the message to be processed in a sequential order, tree constructions split the message into blocks which could be processed by independent processors or machines and the final result is combined to produce the hash value. For applications where large amounts of data have to be hashed using parallel processors, tree constructions are much more appropriate. They have the disadvantage of a larger state information that needs to be kept (logarithmic in the message length, as opposed to linear), but have the advantage that different branches in the tree can be computed independently.

In this section, we will discuss three tree-based hash functions, namely the (strengthened) Merkle tree [57], tree hash [48] and XOR-tree hash [48]. For conciceness, we will not discuss variants of these that appeared in literature, such as [58-60], and constructions based on concrete designs, such as [38]. Finally, we mention that Dodis et al. [61] and Bertoni et al. [62] analyzed the required properties of tree hash functions to obtain indifferentiable designs. We note that these results do not apply to the three tree-based designs discussed here.

Strengthened Merkle tree. The strengthened Merkle tree [57] ($\mathcal{SMT}$) is a narrow-pipe, keyless domain extender. Firstly, to hash a message of arbitrary size, it is padded to the correct length with a single 1 bit and a sufficient number of zeroes, such that it fills the minimal number of blocks to produce a message with number of blocks multiple to a^d where a is the arity or number of inputs to the compression function and d is the minimal tree depth required for hashing the message. On the very top level of the tree the message blocks are hashed independently and the intermediate hash values are the result of the second layer of inputs in the tree. The process is iterated until the root of the tree is reached where the final compression function takes

an extra input which is the message length encoding as part of the strengthening.

Damgård [4] and later Andreeva et al. [22] showed that tree hash preserves Coll security, but Sec and Pre are not preserved. No indifferentiability results are known for tree hashes.

Tree hash function. The tree hash [48] ($\mathcal{TH}$) is a narrow-pipe, keyed hash function. It differs from the strengthened Merkle tree-based construction in the sense that an independent key K_i is assigned to each level i of the tree.

For equal-length messages (thus in case $\mathcal{TH}$ is a fixed input length domain extender), the design has been showed eSec secure by Bellare and Rogaway [48], but in the general case (where the target and forged message may be of different lengths), the domain extender is shown to preserve ePre only [22].

XOR-tree hash function. The XOR-tree hash function [48] ($\mathcal{XTH}$) is a narrow-pipe, keyed hash function. Differing from $\mathcal{TH}$, the outputs of every tree level are XORed with a per level key. The number of keys are logarithmic in the size of the padded message in blocks. Several variants of XOR-tree hash function have appeared in the literature [58-60]. These variants work towards minimizing the key schedule for the basic XOR-tree hash while retaining the same security strength. The optimizations are mainly achieved following a key schedule similar to the Shoup's hash function key scheduling scheme.

For equal-length messages, again Bellare and Rogaway [48] showed this domain extender to preserve eSec. For arbitrary-length messages, Andreeva et al. [22] showed that this construction preserves Pre and ePre, and that it does not preserve aSec and aPre. The authors were unable to show or contradict the preservation of Coll and Sec, no to show (non)-preservation of eSec for arbitrary-length messages.

To fix the deficiencies of the basic XOR-tree hash function for arbitrary message lengths with respect to property preservation, Andreeva et al. [63] suggest a slightly modified construction called the modified XOR-tree hash function. The construction is proved to preserve all three notions of Coll, Sec, and Pre. The only difference with the original XOR-tree domain extender is that here, the key used before the final application of the compression function is a fixed key K^*, which means that K^* is independent of the tree depth, while in the original XOR-tree extender it would simply be the next key in the sequence.

6.2 Multi-Pass Hash Functions

A *multi-pass* domain extender processes the data in more than one pass. Multi-pass designs can process additionally interleaved parts of the message. The idea behind having less efficient multi-pipe designs is to ensure better security guarantees. However, as becomes clear below, this goal is not always so easily attained.

Double pipe design. The double-pipe design [24] ($\mathcal{LDP}$) was proposed by Lucks. It processes the message in two dependent Merkle-Damgård chains under two distinct initialization vectors and the final result is computed by applying a final output transformation, which is not necessarily distinct from the internal chained compression function(s).

Lucks double-pipe hash function achieves the same preservation security guarantees as the Merkle-Damgård design. In particular, the counterexamples for the preservation of Sec and

Pre of the Merkle-Damgård design as illustrated in [22], apply to $\mathcal{LDP}$ similarly. Collision resistance of $\mathcal{LDP}$ follows from the collision preservation proof of [39]: it can be shown that collisions for the domain extender $\mathcal{LDP}$ can be reduced to collisions for either F, or G defined as $\mathrm{G}(h,h',m) = (\mathrm{F}(h \,\|\, h' \,\|\, m), \mathrm{F}(h' \,\|\, h \,\|\, m))$, but it is clear that collisions for G and F are equally hard to find. Finally, the $\mathcal{LDP}$ domain extender is proven indifferentiable in [64].

Zipper hash function. The zipper hash [65] ($\mathcal{Zipper}$) was proposed by Liskov. The message is processed forward in a Merkle-Damgård fashion. Then, the intermediate hash result is used to initialize the further iterative processing of the same message in a reverse order (reversing the data blocks), possibly with some distinct underlying compression function under the Merkle-Damgård iteration.

The counterexamples for the Merkle-Damgård hash of Andreeva et al. apply to the zipper hash to show that no Sec and Pre preservation is achievable for zipper hash. Additionally, no security results are known on the Coll preservation of $\mathcal{Zipper}$. The zipper hash is proven indifferentiable from a random oracle up to the birthday bound even if the compression functions in use are weak, i.e., they can be inverted and collisions can be found efficiently.

Concatenated hash function. The concatenated hash function is a *multi-pipe* design in the sense that it concatenates the hash result of the processing of multiple independent message pipes. Each pipe produces a hash result using a domain extension method. The final hash value is the concatenation of the hash values from all message pipes. The concatenated hash is therefore a general transform that is applied as a form of final transformation on the result of hash values of independent evaluations of the same message under identical or distinct domain extenders. The most popular in the literature is the concatenation of multiple pipes of Merkle-Damgård chains, each initialized under distinct initialization vectors IV s. In general, concatenated hashing can be applied to any combination of domain extenders, and therefore we do not include it in Fig. 1 and Table 1.

It was broadly believed that the bit security level of concatenated hash scales with the number of hash pipes employed. For iterated domain extenders this turns out not to be the case as collision attacks of Joux [12] show where the security level is only marginally higher than a single plain Merkle-Damgård chain. As for the preservation property results, we notice that these follow from the combined security of the underlying domain extenders.

7. Block Cipher or Permutation Based Domain Extenders

A common approach to design compression functions is by building them based on one or more block ciphers or permutations, and we will briefly elaborate on it. Notice, however, that the security notions (Sect. 2.2) are not relevant to these primitives. In particular, collisions for a permutation do not exist, and additionally, block ciphers and permutations are usually easily invertible. As a consequence, when proving security properties of block cipher or permutation based compression functions, one usually relies on the *ideal model*. In the ideal model, the underlying building blocks are assumed to behave like random primitives, and the adversary has only query access to these primitives. The adversary is computationally unbounded, and his advantage is measured in the number of queries made to the oracles. Note that if a compression

function F is atk-secure in the ideal model, and a domain extender is applied on top of F that is atk-preserving in the sense of Sect. 2.2 (Table 1), then the obtained hash function can be considered atk-secure in the ideal model as well.

Consequently, there exist two ways to obtain security results of hash functions based on permutations or block ciphers. First, if a compression function based on a block cipher or permutation is proven secure in the ideal cipher model, one can exhibit further preservation of that property by the domain extender (Sects. 3 and 5) again in the ideal cipher model. Alternatively, if the compression function built on permutation(s) or block cipher(s) is trivially insecure with respect to the basic properties, then one aims at proving direct security of the domain extender assuming ideal behavior of the underlying primitive (permutation or block cipher).

7.1 Block Cipher Based Domain Extenders

A block cipher $E: \mathcal{K} \times \mathcal{M} \to \mathcal{M}$ is a family of permutations indexed by $\mathcal{K}$, and associated to it is its inverse function $E^{-1}: \mathcal{K} \times \mathcal{M} \to \mathcal{M}$. Usually, both E, E^{-1} are efficiently computable, and the adversary has access to both.

Compression function security. Preneel, Govaerts and Vandewalle (PGV) [66] analyzed and categorized 64 block cipher based compression functions, and twelve of them are formally proven collision and preimage resistant by Black, Rogaway and Shrimpton [67]. Among these are the Matyas-Meyer-Oseas, the Miyaguchi-Preneel and the Davies-Meyer compression functions. We note that all of these compression functions are keyless by design, and as such the always and everywhere second preimage and preimage security is inapplicable.

Additionally, we note that the twelve secure PGV constructions are insecure in the indifferentiability model, merely due to the presence of fixed-points [66, 68]. The results of [66, 67] have been recently generalized by Stam [69], to cover more general block cipher based compression function designs.

Security of the domain extender. Additionally to the results on the compression function, Black et al. showed that eight of the remaining 52 PGV compression functions, would result in collision and (non-optimal) preimage security in the $\mathcal{SMD}$ iteration. These bounds have been improved by Duo and Li [70]. Additionally, Duo and Li analyze the second preimage resistance of the 20 PGV-based hash function designs.

This result is, again, generalized by Stam [69]. The way to prove security of this form is generally done in a graph-based approach, where the edge correspond to compression function executions that can be realized using the queries to the oracle made by the adversary. It is shown that all of twenty PGV compression functions are indifferentiable from a random oracle when iterated in a $\mathcal{MD}$ with chop, NMAC and HMAC construction, and for sixteen of the twenty constructions, the $\mathcal{PfMD}$uction results in an indifferentiable design [11, 42, 71, 72].

7.2 Permutation Based Domain Extenders

A permutation $\pi: \mathcal{M} \to \mathcal{M}$ is a map for which the domain and range space are identical. Therefore, in order to build a permutation-based compression function, the compressing needs to happen at a different time in the execution (e.g. before the permutation π is executed).

Compression function security. In [73], Black, Cochran and Shrimpton analyzed $2n$-bit to n-bit compression functions based on one n-bit permutation, and proved them insecure against collision and (second) preimage attacks. This result has been generalized by Rogaway and Steinberger [74] and Stam [75] to compression functions with arbitrary input and output sizes, and an arbitrary number of underlying permutations. Their bounds indicate the expected number of queries required to find collisions or preimages for permutation based compression functions. As positive results, Rogaway and Steinberger [76] and Shrimpton and Stam [77] propose $2n$- to n-bit compression functions based on 3 permutations that achieve optimal collision and preimage resistance with respect to the bounds of [74, 75]. No results are known on the second preimage resistance. No positive indifferentiability results are known for permutation based compression functions.

Security of the domain extender. In the iteration, permutation based compression functions may result in a secure hash function, though. A well-known example of this is the sponge construction, due to Bertoni et al. [78]. A broad interpretation of the sponge hash function renders it a keyless, wide-pipe, non-strengthened Merkle-Damgård construction. The sponge hash function iterates a state size of $c+r$ bits, where r is the bit rate and c is the capacity of the sponge. It consists of an absorbing phase and a squeezing phase. In the absorbing phase, message blocks of r bits are compressed with the state by ways of its compression function $\mathrm{F}(h,m) = \pi(h \oplus (m \,\|\, 0^c))$. After the processing of all message blocks is completed, a final transformation is applied, the squeezing phase, where the first r bits of the state are returned as output blocks, interleaved with applications of the permutation until the desired output length is achieved. Notice, that in contrast to other domain extenders, the sponge hash function supports by design arbitrary output lengths. A well-known sponge construction is Keccak by Bertoni et al. [31].

Obviously, the sponge compression function is vulnerable to collision, second preimage and preimage attacks [73], and we cannot rely on the preservation strength of the design to obtain security properties of the hash function (in the ideal model). Yet, the sponge construction is proven indifferentiable from a random oracle if the permutation π is assumed to be ideal [79]. In the ideal model, this results in collision, second preimage and preimage security of the sponge design [39].

Several “sponge-like” designs are known in literature, among which the Grindahl design by Knudsen, Rechberger and Thomsen [80], that do not exactly follow the original sponge design. Similar approaches can be applied to these domain extenders to obtain similar security bounds.

8. Application to NIST’s SHA-3 Competition

We will briefly consider the application of the classification of this work to NIST's SHA-3 competition. We refer to [39] for a more detailed discussion of the security properties of the 14 second round SHA-3 candidates.

First of all, each of the 14 SHA-3 candidates can be seen as an extension of the keyless Merkle-Damgård design, with an optional final transformation and/or chopping. Five of the designs employ a prefix-free padding rule, and can be seen as $\mathcal{PfMD}$ constructions (three of which moreover fit within the $\mathcal{HAIFA}$ design). Eleven of the designs have a suffix-free padding

rule, and therefore preserve collision resistance (Sect. 4.2). No preservation results are known for the remaining security properties.

Refining the level of modularity, the SHA-3 candidates SHAvite-3 and Skein employ a PGV-construction proven collision and preimage secure in [67], and the compression functions of ECHO, Hamsi and SIMD fit in the generalized block cipher based model of [69]. For all of these designs, their domain extenders preserve collision resistance, which renders collision resistance bounds of the hash functions in the ideal model. The compression function of the SHA-3 candidate Grøstl can be proven preimage and collision resistant up to the bound of [74]. The Grøstl domain extender preserves collision resistance, but not preimage resistance.

The SHA-3 candidates CubeHash, Fugue, JH, Keccak and Luffa can be considered sponge(-like) designs, and all of these functions have a compression function vulnerable to the attacks described in [73]. The indifferentiability result of [79] can be directly applied to CubeHash and Keccak.

We notice that for some of the candidates, design-specific security approaches have resulted in other security results [39], in particular with respect to indifferentiability bounds. Finally, we notice that indifferentiability bounds imply an upper bound on the success probabilities of breaking a hash function under any security notion [39]. Namely, one can show that for any security notion atk of a hash function in the ideal model (Sect. 7), we have $\mathbf{Adv}^{\mathrm{atk}}_{\mathrm{H}}(q) \leq \mathbf{Pr}^{\mathrm{atk}}_{\mathrm{RO}}(q) + \mathbf{Adv}^{\mathrm{PRO}}_{\mathrm{H}}(q)$, where $\mathbf{Adv}^{\mathrm{atk}}_{\mathrm{H}}(q)$ denotes the maximum advantage of any adversary/distinguisher against security notion atk making at most q queries, and $\mathbf{Pr}^{\mathrm{atk}}_{\mathrm{RO}}(q)$ denotes the success probability of a generic attack against $\mathcal{H}$ under atk, after at most q queries.

For some of the SHA-3 candidates, using this result the indifferentiability bounds have resulted in security bounds on the preimage, second preimage and collision resistance (in the ideal model). It is clear from the above that optimal preimage, second preimage and collision resistance is obtained if $\mathbf{Adv}^{\mathrm{PRO}}_{\mathrm{H}}(q) = O(q / 2^n)$.

9. CONCLUSION

We analyzed the state-of-the-art security results of the original Merkle-Damgård design and its derivatives, with respect to the security definitions posited by Rogaway and Shrimpton [15], and to the notion of pseudorandom oracle behavior, as formalized by Coron et al. in the context of hash functions [11]. These result consider the security preservation properties of the domain extenders from the underlying compression functions, and intuitively mean that if an attacker can break a hash function under a security notion atk, then, one can reduce this attack to an atk-forgery of the compression function. The presented analysis extends the work of [22] in the sense that a wider variety of domain extenders is considered, and that the security property PRO (pseudorandom oracle behavior) is considered as well. A summary of the results is given in Table 1. Most of the 17 domain extenders considered in this work preserve collision resistance, and about half of them is provably indifferentiable from a random oracle. On the downside, (second) preimage is only preserved by a small fraction of domain extenders. A way to resolve this preservation property issue is given in Sect. 4.2, but this still leaves a further direction in the area of provable security.

Besides, we briefly summarized the state-of-the-art security results of hash functions of which the underlying compression function consists of a block cipher or one or more permutations, and applied these results to the NIST SHA-3 hash function competition [16]. This survey partially summarizes the classification by Andreeva, Mennink and Preneel [39]. Here, one does not consider "preservation" of security properties, but rather the security is considered in case the underlying primitives are assumed to be ideal (for instance, a random permutation or random block cipher).

REFERENCES

[1] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644-654, 1976.

[2] Ralph Merkle. *Secrecy, Authentication, and Public Key Systems*. UMI Research Press, 1979.

[3] Michael O. Rabin. Digitalized signatures. *Foundations of Secure Computation*, New York, 1978. Academic Press, pp.155-166.

[4] Ivan Damgård. A Design Principle for Hash Functions. *Advances in Cryptology - CRYPTO '89*, Vol.435 of Lecture Notes in Computer Science, Berlin, 1990. Springer-Verlag, pp.416-427.

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

[6] Xuejia Lai and James Massey. Hash Function Based on Block Ciphers. *Advances in Cryptology - EUROCRYPT '92*, Vol.658 of Lecture Notes in Computer Science, Berlin, 1992. Springer-Verlag, pp.55-70.

[7] Bert den Boer and Antoon Bosselaers. Collisions for the compression function of MD5. *Advances in Cryptology - EUROCRYPT '93*, Vol.765 of Lecture Notes in Computer Science, Berlin, 1994. Springer-Verlag, pp.293-304.

[8] Hans Dobbertin. The status of MD5 after a recent attack. *CryptoBytes*, 2(2):1-6, 1996.

[9] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. *Advances in Cryptology - EUROCRYPT '05*, Vol.3494 of Lecture Notes in Computer Science, Berlin, 2005. Springer-Verlag, pp.19-35.

[10] Xiaoyun Wang and Yiqun Lisa Yin and Hongbo Yu. Finding Collisions in the Full SHA-1. *Advances in Cryptology - CRYPTO '05*, Vol.3621 of Lecture Notes in Computer Science, Berlin, 2005. Springer-Verlag, pp.17-36.

[11] Jean-Sébastien Coron and Yevgeniy Dodis and Cécile Malinaud and Prashant Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. *Advances in Cryptology - CRYPTO '05*, Vol.3621 of Lecture Notes in Computer Science, Berlin, 2005. Springer-Verlag, pp.430-448.

[12] Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. *Advances in Cryptology - CRYPTO '04*, Vol.3152 of Lecture Notes in Computer Science, Berlin, 2004. Springer-Verlag, pp.306-316.

[13] John Kelsey and Tadayoshi Kohno. Herding Hash Functions and the Nostradamus Attack. *Advances in Cryptology - EUROCRYPT'06*, Vol.4004 of Lecture Notes in Computer Science, Berlin, 2006. Springer-Verlag, pp.183-200.

[14] John Kelsey and Bruce Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. *Advances in Cryptology - EUROCRYPT '05*, Vol.3494 of Lecture Notes in Computer Science, Berlin, 2005. Springer-Verlag, pp.474-490.

[15] Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. *Fast Software Encryption '04*, Vol.3017 of Lecture Notes in Computer Science, Berlin, 2004. Springer-Verlag, pp.371-388.

[16] National Institute for Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. 2007. http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf.

[17] Douglas Stinson. Some Observations on the Theory of Cryptographic Hash Functions. *Des. Codes Cryptography*, 38(2):259-277, 2006.

[18] Phillip Rogaway. Formalizing Human Ignorance. *VIETCRYPT '92*, Vol.4341 of Lecture Notes in Computer Science, Berlin, 2006. Springer-Verlag, pp.211-228.

[19] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. *Advances in Cryptology - CRYPTO '86*, Vol.263 of Lecture Notes in Computer Science, Berlin, 1987. Springer-Verlag, pp.186-194.

[20] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. *ACM Conference on Computer and Communications Security*, New York, 1993. ACM, pp.62-73.

[21] Ueli Maurer and Renato Renner and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. *Theory of Cryptography Conference '04*, Vol.2951 of Lecture Notes in Computer Science, Berlin, 2004. Springer-Verlag, pp.21-39.

[22] Elena Andreeva and Gregory Neven and Bart Preneel and Thomas Shrimpton. Seven-Property-Preserving Iterated Hashing: ROX. *Advances in Cryptology - ASIACRYPT '07*, Vol.4833 of Lecture Notes in Computer Science, Berlin, 2007. Springer-Verlag, pp.130-146.

[23] Richard Dean. *Formal Aspects of Mobile Code Security*. PhD thesis, Princeton University, Princeton, 1999.

[24] Stefan Lucks. A Failure-Friendly Design Principle for Hash Functions. *Advances in Cryptology - ASIACRYPT '05*, Vol.3788 of Lecture Notes in Computer Science, Berlin, 2005. Springer-Verlag, pp.474-494.

[25] Hans Danilo Gligoroski and Vlastimil Klima and Svein Johan Knapskog and Mohamed El-Hadedy and Jørn Amundsen and Stig Frode Mjølsnes. Cryptographic Hash Function BLUE MIDNIGHT WISH. 2009.

[26] Dan Bernstein. CubeHash specification. 2009.

[27] Ryad Benadjila and Olivier Billet and Henri Gilbert and Gilles Macario-Rat and Thomas Peyrin and Matt Robshaw and Yannick Seurin. SHA-3 Proposal: ECHO. 2009.

[28] Shai Halevi and William Hall and Charanjit Jutla. The Hash Function "Fugue". 2009.

[29] Praveen Gauravaram and Lars Knudsen and Krystian Matusiewicz and Florian Mendel and Christian Rechberger and Martin Schläffer and Søren Thomsen. Grøstl - a SHA-3 candidate. 2009.

[30] Hongjun Wu. The Hash Function JH. 2009.

[31] Guido Bertoni and Joan Daemen and Michaël Peeters and Gilles Van Assche. The KECCAK sponge function family. 2009.

[32] Christophe De Cannière and Hisayoshi Sato and Dai Watanabe. Hash Function Luffa. 2009.

[33] Gaëtan Leurent and Charles Bouillaguet and Pierre-Alain Fouque. SIMD is a Message Digest. 2009.

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

[35] Jean-Philippe Aumasson and Luca Henzen and Willi Meier and Raphael Phan. SHA-3 proposal BLAKE. 2009.

[36] Özgül Küçük. The Hash Function Hamsi. 2009.

[37] Eli Biham and Orr Dunkelman. The SHAvite-3 Hash Function. 2009.

[38] Shai Niels Ferguson and Stefan Lucks and Bruce Schneier and Doug Whiting and Mihir Bellare and Tadayoshi Kohno and Jon Callas and Jesse Walker. The Skein Hash Function Family. 2009.

[39] Elena Andreeva and Bart Mennink and Bart Preneel. Security Reductions of the Second Round SHA-3 Candidates. *Information Security Conference - ISC '10* in Lecture Notes in Computer Science, Berlin, 2010. Springer-Verlag. To appear.

[40] Mihir Bellare and Thomas Ristenpart. Multi-Property-Preserving Hash Domain Extension and the

EMD Transform. *Advances in Cryptology - ASIACRYPT '06*, Vol.4284 of Lecture Notes in Computer Science, Berlin, 2006. Springer-Verlag, pp.299-314.

[41] Elena Andreeva and Bart Preneel. A Three-Property-Secure Hash Function. *Selected Areas in Cryptography '08*, Vol.5381 of Lecture Notes in Computer Science, Berlin, 2009. Springer-Verlag, pp.228-244.

[42] Donghoon Chang and Sangjin Lee and Mridul Nandi and Moti Yung. Indifferentiable Security Analysis of Popular Hash Functions with Prefix-Free Padding. *Advances in Cryptology - ASIACRYPT '06*, Vol.4284 of Lecture Notes in Computer Science, Berlin, 2006. Springer-Verlag, pp.283-298.

[43] Donghoon Chang and Mridul Nandi. Improved Indifferentiability Security Analysis of chopMD Hash Function. *Fast Software Encryption '08*, Vol.5086 of Lecture Notes in Computer Science, Berlin, 2008. Springer-Verlag, pp.429-443.

[44] Ewan Fleischmann and Michael Gorski and Stefan Lucks. Some Observations on Indifferentiability. *Australasian Conference on Information Security and Privacy - ACISP '10*, Vol.6168 of Lecture Notes in Computer Science, Berlin, 2010. Springer-Verlag, pp.117-134.

[45] Yevgeniy Dodis and Thomas Ristenpart and Thomas Shrimpton. Salvaging Merkle-Damgård for Practical Applications. *Advances in Cryptology - EUROCRYPT '09*, Vol.5479 of Lecture Notes in Computer Science, Berlin, 2009. Springer-Verlag, pp.371-388.

[46] Mihir Bellare and Ran Canetti and Hugo Krawczyk. Keying Hash Functions for Message Authentication. *Advances in Cryptology - CRYPTO '96*, Vol.1109 of Lecture Notes in Computer Science, Berlin, 1996. Springer-Verlag, pp.1-15.

[47] Shoichi Hirose and Je Hong Park and Aaram Yun. A Simple Variant of the Merkle-Damgård Scheme with a Permutation. *Advances in Cryptology - ASIACRYPT '07*, Vol.4833 of Lecture Notes in Computer Science, Berlin, 2007. Springer-Verlag, pp.113-129.

[48] Mihir Bellare and Phillip Rogaway. Collision-Resistant Hashing: Towards Making UOWHFs Practical. *Advances in Cryptology - CRYPTO '97*, Vol.1294 of Lecture Notes in Computer Science, Berlin, 1997. Springer-Verlag, pp.470-484.

[49] Mihir Bellare and Thomas Ristenpart. Hash Functions in the Dedicated-Key Setting: Design Choices and MPP Transforms. *International Colloquium on Automata, Languages and Programming - ICALP '07*, Vol.4596 of Lecture Notes in Computer Science, Berlin, 2007. Springer-Verlag, pp.399-410.

[50] Victor Shoup. A Composition Theorem for Universal One-Way Hash Functions. *Advances in Cryptology - EUROCRYPT '00*, Vol.1807 of Lecture Notes in Computer Science, Berlin, 2000. Springer-Verlag, pp.445-452.

[51] Mohammad Reza Reyhanitabar and Willy Susilo and Yi Mu. Analysis of Property-Preservation Capabilities of the ROX and ESh Hash Domain Extenders. *Australasian Conference on Information Security and Privacy - ACISP '09*, Vol.5594 of Lecture Notes in Computer Science, Berlin, 2009. Springer-Verlag, pp.153-170.

[52] Eli Biham and Orr Dunkelman. A Framework for Iterative Hash Functions - HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007.

[53] Charles Bouillaguet and Pierre-Alain Fouque and Adi Shamir and Sébastien Zimmer. Second Preimage Attacks on Dithered Hash Functions. Cryptology ePrint Archive, Report 2007/395, 2007.

[54] Ronald Rivest. Abelian Square-Free Dithering for Iterated Hash Functions. ECRYPT Hash Function Workshop 2005.

[55] Shai Halevi and Hugo Krawczyk. Strengthening Digital Signatures Via Randomized Hashing. *Advances in Cryptology - CRYPTO '06*, Vol.4117 of Lecture Notes in Computer Science, Berlin, 2006. Springer-Verlag, pp.41-59.

[56] Praveen Gauravaram and Lars Knudsen. On Randomizing Hash Functions to Strengthen the Security of Digital Signatures. *Advances in Cryptology - EUROCRYPT '09*, Vol.5479 of Lecture Notes in Computer Science, Berlin, 2009. Springer-Verlag, pp.88-105.

[57] Ralph Merkle. Protocols for Public Key Cryptosystems. *IEEE Symposium on Security and Privacy*, 1980. IEEE Computer Society Press, pp.122-134.

[58] Wonil Lee and Donghoon Chang and Sangjin Lee and Soo Hak Sung and Mridul Nandi. New Parallel Domain Extenders for UOWHF. *Advances in Cryptology - ASIACRYPT '03*, Vol.2894 of Lecture

Notes in Computer Science, Berlin, 2003. Springer-Verlag, pp.208-227.

[59] Wonil Lee and Donghoon Chang and Sangjin Lee and Soo Hak Sung and Mridul Nandi. Construction of UOWHF: Two New Parallel Methods. *IEICE Transactions*, 88-A(1):49-58, 2005.

[60] Palash Sarkar. Construction of Universal One-Way Hash Functions: Tree Hashing Revisited. *Discrete Applied Mathematics*, 155(16):2174-2180, 2007.

[61] Yevgeniy Dodis and Leonid Reyzin and Ronald Rivest and Emily Shen. Indifferentiability of Permutation-Based Compression Functions and Tree-Based Modes of Operation, with Applications to MD6. *Fast Software Encryption '09*, Vol.5665 of Lecture Notes in Computer Science, Berlin, 2009. Springer-Verlag, pp.104-121.

[62] Guido Bertoni and Joan Daemen and Michaël Peeters and Gilles van Assche. Sufficient Conditions for Sound Tree and Sequential Hashing Modes. Cryptology ePrint Archive, Report 2009/210, 2009.

[63] Elena Andreeva and Gregory Neven and Bart Preneel and Thomas Shrimpton. Three-Property-Preserving Iterations of Keyless Compression Functions. ECRYPT Hash Function Workshop 2007.

[64] Jonathan Hoch and Adi Shamir. On the Strength of the Concatenated Hash Combiner When All the Hash Functions Are Weak. *International Colloquium on Automata, Languages and Programming - ICALP '08*, Vol.5126 of Lecture Notes in Computer Science, Berlin, 2008. Springer-Verlag, pp.616-630.

[65] Moses Liskov. Constructing an Ideal Hash Function from Weak Ideal Compression Functions. *Selected Areas in Cryptography '06*, Vol.4356 of Lecture Notes in Computer Science, Berlin, 2007. Springer-Verlag, pp.358-375.

[66] Bart Preneel and René Govaerts and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. *Advances in Cryptology - CRYPTO '93*, Vol.773 of Lecture Notes in Computer Science, Berlin, 1993. Springer-Verlag, pp.368-378.

[67] John Black and Phillip Rogaway and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. *Advances in Cryptology - CRYPTO '02*, Vol.2442 of Lecture Notes in Computer Science, Berlin, 2002. Springer-Verlag, pp.320-335.

[68] Hidenori Kuwakado and Masakatu Morii. Indifferentiability of Single-Block-Length and Rate-1 Compression Functions. *IEICE Transactions*, 90-A(10):2301-2308, 2007.

[69] Martijn Stam. Blockcipher-Based Hashing Revisited. *Fast Software Encryption '09*, Vol.5665 of Lecture Notes in Computer Science, Berlin, 2009. Springer-Verlag, pp.67-83.

[70] Lei Duo and Chao Li. Improved Collision and Preimage Resistance Bounds on PGV Schemes. Cryptology ePrint Archive, Report 2006/462, 2006.

[71] Zheng Gong and Xuejia Lai and Kefei Chen. A Synthetic Indifferentiability Analysis of Some Block-Cipher-Based Hash Functions. *Des. Codes Cryptography*, 48(3):293-305, 2008.

[72] Yiyuan Luo and Zheng Gong and Ming Duan and Bo Zhu and Xuejia Lai. Revisiting the Indifferentiability of PGV Hash Functions. Cryptology ePrint Archive, Report 2009/265, 2009

[73] John Black and Martin Cochran and Thomas Shrimpton. On the Impossibility of Highly-Efficient Blockcipher-Based Hash Functions. *Advances in Cryptology - EUROCRYPT '05*, Vol.3494 of Lecture Notes in Computer Science, Berlin, 2005. Springer-Verlag, pp.526-541.

[74] Phillip Rogaway and John Steinberger. Security/Efficiency Tradeoffs for Permutation-Based Hashing. *Advances in Cryptology - EUROCRYPT '08*, Vol.4965 of Lecture Notes in Computer Science, Berlin, 2008. Springer-Verlag, pp.220-236.

[75] Martijn Stam. Beyond Uniformity: Better Security/Efficiency Tradeoffs for Compression Functions. *Advances in Cryptology - CRYPTO '08*, Vol.5157 of Lecture Notes in Computer Science, Berlin, 2008. Springer-Verlag, pp.397-412.

[76] Phillip Rogaway and John Steinberger. Constructing Cryptographic Hash Functions from Fixed-Key Blockciphers. *Advances in Cryptology - CRYPTO '08*, Vol.5157 of Lecture Notes in Computer Science, Berlin, 2008. Springer-Verlag, pp.433-450.

[77] Thomas Shrimpton and Martijn Stam. Building a Collision-Resistant Compression Function from Non-compressing Primitives. *International Colloquium on Automata, Languages and Programming - ICALP '08*, Vol.5126 of Lecture Notes in Computer Science, Berlin, 2008. Springer-Verlag, pp.643-654.

[78] Guido Bertoni and Joan Daemen and Michaël Peeters and Gilles Van Assche. Sponge Functions. ECRYPT Hash Function Workshop 2007.

[79] Guido Bertoni and Joan Daemen and Michaël Peeters and Gilles van Assche. On the Indifferentiability of the Sponge Construction. *Advances in Cryptology - EUROCRYPT '08*, Vol.4965 of Lecture Notes in Computer Science, Berlin, 2008. Springer-Verlag, pp.181-197.

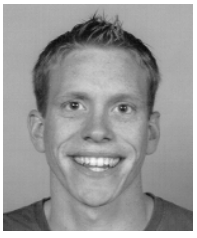
[80] Lars Knudsen and Christian Rechberger and Søren Thomsen. The Grindahl Hash Functions. *Fast Software Encryption '07*, Vol.4593 of Lecture Notes in Computer Science, Berlin, 2007. Springer-Verlag, pp.39-57.

[81] Yevgeniy Dodis and Prashant Puniya. Getting the Best Out of Existing Hash Functions; or What if We Are Stuck with SHA?. *Applied Cryptography and Network Security '08*, Vol.5037 of Lecture Notes in Computer Science, Berlin, 2008. Springer-Verlag, pp.156-173.

[82] Mridul Nandi. Characterizing Padding Rules of MD Hash Functions Preserving Collision Security. *Australasian Conference on Information Security and Privacy '09*, Vol.5594 of Lecture Notes in Computer Science, Berlin, 2009. Springer-Verlag, pp.171-184.

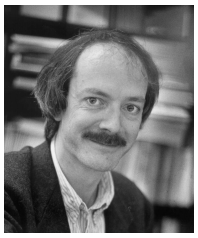
Elena Andreeva

She received her bachelor degree in Computer Science from the University of Varna, Bulgaria, and her master's degree in Computer Science from the Saaland University, Germany. She completed her Ph.D. in Cryptography on the topic of "Domain Extenders for Cryptographic Hash Functions" from the Katholieke Universiteit Leuven under the supervision of Prof. Bart Preneel. From October 2007, her PhD research was sponsored by the Fund for Scientific Research, Flanders (FWO-Vlaanderen). Her research interests include symmetric primitives and provable security.



Bart Mennink

He received his bachelor and master degree in Mathematics from the Technische Universiteit Eindhoven, The Netherlands, in 2007 and 2009, respectively. Currently he is a Ph.D. candidate at the Katholieke Universiteit Leuven, Belgium, under the supervision of Prof. Bart Preneel and Prof. Vincent Rijmen. He is funded by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen). His research interests include symmetric primitives and provable security.



Bart Preneel

He received the M.S. degree in electrical engineering and the Ph.D. degree in applied sciences (cryptology) from the Katholieke Universiteit Leuven, Belgium, in 1987 and 1993, respectively. He is currently Full Professor with the Katholieke Universiteit Leuven. He was Visiting Professor at five universities in Europe and was a Research Fellow with the University of California at Berkeley. He has authored and coauthored more than 300 reviewed scientific publications and is the inventor of three patents. His main research interests are cryptography and information security. Prof. Preneel is President of the International Association for Cryptologic Research (IACR) and of the Leuven Security Excellence Consortium (L-SEC vzw.), an association of 60 companies and research institutions in the area of e-security.

A. PRESERVATION OF COLLISION RESISTANCE

In this appendix, we generalize the well-known collision resistance preservation result of the strengthened Merkle-Damgård design [5, 4]. The result of Thm. 1 differs in three cases: we consider any suffix-free padding, the proof allows for different compression functions in one hash function evaluation, and it includes an optional chopping at the end. Related work can, a.o., be found in [5, 4, 81, 82].

Theorem 1. *Let* $l, b, n \in \mathbb{N}$ *such that* $l \geq n$. *Let* $\mathrm{pad}: \{0,1\}^* \to \{0,1\}^{b*}$ *be a suffix-free padding and let* $f: \{0,1\}^l \times \{0,1\}^b \to \{0,1\}^l$ *and* $g: \{0,1\}^l \times \{0,1\}^b \to \{0,1\}^n$ *be two compression functions. Consider the hash function* $\mathcal{H}: \{0,1\}^* \to \{0,1\}^n$ *defined as follows, where* $h_0 = IV$ *is the initialization vector:*

$$\begin{aligned} \mathcal{H}(M) = h_k, \text{ where: } & (M_1, \ldots, M_k) = \mathrm{pad}(M), \\ & h_i = f(h_{i-1}, M_i) \text{ for } i = 1, \ldots, k-1, \\ & h_k = g(h_{k-1}, M_k). \end{aligned}$$

The advantage of finding collisions for $\mathcal{H}$ *is upper bounded by the advantage of finding collisions for* f *or* g. *Formally, if* f *is* (t_1, ε_1) *collision secure, and* g *is* (t_2, ε_2) *collision secure, then* $\mathcal{H}$ *is* (t, ε) *collision secure for* $\varepsilon = \varepsilon_1 + \varepsilon_2$, *and* $t = \min\{t_1, t_2\} - 2(K-1)\tau_f$, *where* τ_f *is the time to evaluate* f *and* K *is the maximum length of the messages, in blocks.*

Proof. Suppose A is a (t, ε) collision finding attacker for $\mathcal{H}$. We construct collision finding adversaries B_1 and B_2 for f and g, respectively, using the following observation.

Let M, M' be two distinct messages such that $\mathcal{H}(M) = \mathcal{H}(M')$. Let $(M_1, \ldots, M_k)$ be the padded message of M, and $(M'_1, \ldots, M'_{k'})$ be the padded message of M'. Define the intermediate state values $h_i, h_{i'}$ similarly. A collision on M, M' means that $g(h_{k-1}, M_k) = g(h_{k'-1}', M_{k'}')$. Now, if $(h_{k-1}, M_k) \neq (h_{k'-1}', M_{k'}')$ this results in a collision for g. Assume the contrary, and let $j \in \{1, \ldots, \min\{k, k'\} - 1\}$ be the minimal index such that $(h_{k-j-1}, M_{k-j}) \neq (h_{k'-j-1}', M_{k'-j}')$. We notice that such index j exists: in case $k = k'$ it exists as $M \neq M'$, and in case $k \neq k'$ it exists as the padding rule is suffix-free. By definition of the index j, we have $h_{k-j} = h_{k'-j}'$, and in particular we obtain a collision for f:

$$f(h_{k-j-1}, M_{k-j}) = h_{k-j} = h_{k'-j}' = f(h_{k'-j-1}', M_{k'-j}').$$

Both $\mathrm{B}_1, \mathrm{B}_2$ follow this procedure. If M, M' define a collision for f, B_1 outputs this collision. Similarly for B_2 and g. Both adversaries work in time at most $t + 2(K-1)\tau_f$, from which we deduce $t \geq \min\{t_1, t_2\} - 2(K-1)\tau_f$. The messages M, M' define a collision for f or g. Thus, we obtain $\varepsilon \leq \varepsilon_1 + \varepsilon_2$. □

The functions f and g in the proof can in general be any function, and in particular need not be independent. For example, if $l = n$ and $g = f$ we retain the original Merkle-Damgård design, and if $g = f \circ \pi$ we obtain the $\mathcal{MDP}$ mode of operation. The Merkle-Damgård design with chopping is also covered, with $g = \mathrm{chop}_{l-n} \circ f$. We note that Thm. 1 can be generalized arbitrarily, e.g. to more different compression functions, but for the purpose of this paper, the mentioned generalization of the Merkle-Damgård structure suffices.

B. Preservation Proofs for Merkle-Damgård with Permutation

In this section, we formally prove that the $\mathcal{MDP}$ domain extender does not preserve second preimage (Sec) and preimage resistance (Pre). This is done by constructing a atk-secure compression function G, for which $\mathcal{MDP}_{\mathrm{G}}$ is *not* atk-secure. The proof is similar to the proofs of [22], but differs in several aspects in order to cover the Merkle-Damgård with permutation construction.

Theorem 2. *For* $\mathrm{atk} \in \{\mathrm{Sec}, \mathrm{Pre}\}$, *the following holds. If there exists a* (t, ε) atk*-secure compression function* $\mathrm{F}: \{0,1\}^{n+b} \to \{0,1\}^{n-2}$, *then there exists a* $(t, \varepsilon + 2/2^n)$ atk*-secure compression function* $\mathrm{G}: \{0,1\}^{n+b} \to \{0,1\}^n$ *and an adversary* A *running in constant time with* $\mathrm{atk}[\lambda]$*-advantage one in breaking* $\mathcal{MDP}_{\mathrm{G}}$.

Proof. Given a compression function $\mathrm{F}: \{0,1\}^{n+b} \to \{0,1\}^{n-2}$, consider the compression function $\mathrm{G}: \{0,1\}^{n+b} \to \{0,1\}^n$ given by:

$$\mathrm{G}(h \| m) = \begin{cases} IV, \text{ if } h = IV \text{ or } h = \pi(IV), \\ \mathrm{F}(h \| m) \| \overline{\pi(IV)}^{(n-1)} \| \overline{IV}^{(n)}, \end{cases}$$

where π is the permutation employed by $\mathcal{MDP}$ (see Fig. 1), and $\overline{x}^{(i)}$ denotes the i^{th} bit of the complement of x.

If F is (t, ε) atk-secure, then G is $(t, \varepsilon + 2/2^n)$ atk-secure. Indeed, for Sec security: given a uniformly random challenge $h \| m \in \{0,1\}^{n+b}$, provided that $h \notin \{IV, \pi(IV)\}$, finding a second preimage is equally hard for F and G. Thus, given an adversary A that breaks G in time t and with probability ε, one can construct an adversary B for F that also runs in time t, who forwards the challenge to A, and simply returns A's response $h' \| m'$. We obtain

$$\begin{aligned} \mathbf{Adv}_{\mathrm{F}}^{\mathrm{Sec}}(\mathrm{B}) &= \Pr\left[\mathrm{G}(h \| m) = \mathrm{G}(h' \| m') \wedge h \notin \{IV, \pi(IV)\}\right] \\ &= \Pr\left[\mathrm{G}(h \| m) = \mathrm{G}(h' \| m')\right] - \Pr\left[h \in \{IV, \pi(IV)\}\right] \\ &= \Pr\left[\mathrm{A} \text{ succeeds}\right] - 2/2^n. \end{aligned}$$

As this holds for any adversary A for G, we obtain $\mathbf{Adv}_{\mathrm{F}}^{\mathrm{Sec}}(\mathrm{B}) = \mathbf{Adv}_{\mathrm{G}}^{\mathrm{Sec}}(\mathrm{A}) - 2/2^n$. The same analysis holds for Pre.

Thus, G is $(t, \varepsilon + 2/2^n)$ atk-secure, but it is clear by construction that $\mathcal{MDP}_{\mathrm{G}}(M) = IV$ for all $M \in \{0,1\}^*$, and hence any message M' renders a (second) preimage. □