Seven-Property-Preserving Iterated Hashing: ROX*

Elena Andreeva[1], Gregory Neven[1,2], Bart Preneel[1], and Thomas Shrimpton[3,4]

[1] SCD-COSIC, Dept. of Electrical Engineering, Katholieke Universiteit Leuven,
{Elena.Andreeva,Gregory.Neven,Bart.Preneel}@esat.kuleuven.be
[2] Département d'Informatique, Ecole Normale Supérieure
[3] Dept. of Computer Science, Portland State University, teshrim@cs.pdx.edu
[4] Faculty of Informatics, University of Lugano

Abstract. Nearly all modern hash functions are constructed by iterating a compression function. At FSE'04, Rogaway and Shrimpton [28] formalized seven security notions for hash functions: collision resistance (Coll) and three variants of second-preimage resistance (Sec, aSec, eSec) and preimage resistance (Pre, aPre, ePre). The main contribution of this paper is in determining, by proof or counterexample, which of these seven notions is preserved by each of eleven existing iterations. Our study points out that none of them preserves more than three notions from [28]. As a second contribution, we propose the new Random-Oracle XOR (ROX) iteration that is the first to provably preserve all seven notions, but that, quite controversially, uses a random oracle in the iteration. The compression function itself is *not* modeled as a random oracle though. Rather, ROX uses an auxiliary small-input random oracle (typically 170 bits) that is called only a logarithmic number of times.

1 Introduction

Cryptographic hash functions, publicly computable maps from inputs of arbitrary length to (short) fixed-length strings, have become a ubiquitous building block in cryptography. Almost all cryptographic hash functions are iterative: given a compression function F that takes $(n + b)$ bits of input and produces n bits of output, they process an arbitrary length input by dividing it into b-bit blocks and iterating F appropriately. The widely used Strengthened Merkle-Damgård (SMD) construction [21, 11] is known to yield a collision-resistant iterated hash function if the underlying compression function is collision resistant; in other words, SMD *preserves* collision resistance of the compression function.

Unfortunately, designing collision resistant compression functions seems quite hard: witness the recent collision attacks on several popular hash functions by Wang et al. [33, 32]. One way out is to aim for a weaker security notion for the compression function, but not so weak as to make the resulting hash function useless in practice. A natural question to ask is whether these weaker properties are also preserved by SMD. For example, does it preserve second-preimage

* Extended abstract; we refer to the full version [1] for more details and proofs.

resistance? One may think so, because SMD preserves collision resistance, and collision resistance can be shown to imply second-preimage resistance, but this says *nothing* about what happens if you *start* with a compression function that is only second-preimage resistant. Lai and Massey [16] claimed that finding second preimages for an iterated hash is equally as hard as finding second preimages for the compression function, but this was found to be incorrect by Dean [12] and Kelsey and Schneier [15], who show that (for the case of SMD) efficient collision-finding attacks immediately give rise to second-preimage attacks that beat the anticipated security bound.

Contributions. We took as a starting point a paper by Rogaway and Shrimpton [28] that provides a unifying framework of seven security notions for hash functions and the relations among them. Our work explores in detail which of the seven properties of [28] are preserved by several published hash constructions. Of the eleven schemes we consider (see Table 1), we found that in fact none preserved all seven. This raises the question whether it is possible at all to preserve all seven properties. We answer this question in the affirmative, in the random oracle model [6], by presenting a construction that builds on previous work by Bellare, Rogaway, Shoup and Mironov [7, 30, 23]. Our construction iterates a *real-world* compression function but, in the iteration, makes a logarithmic (in the message length) number of calls to an auxiliary small-input random oracle; we will say more in a moment to justify this choice. The existence of seven-property-preserving iterations in the standard model is left as an open problem.

Relevance of the seven properties. Apart from collision-resistance, Rogaway and Shrimpton consider three variants of second-preimage resistance (Sec) and preimage resistance (Pre). The standard variants of Sec and Pre are restricted to randomly chosen preimages, and have important applications like the Cramer-Shoup cryptosystem [10] for Sec and Unix-like password storage [18, 31] for Pre. The stronger *everywhere* variants (eSec, ePre) consider adversarially chosen preimages. The notion of eSec is equivalent to the universal one-way hash functions of Naor and Yung [25] and to the target collision resistance of Bellare and Rogaway [7]. Bellare and Rogaway show that eSec is sufficient to extend the message space of signature schemes that are defined for small messages only.

Following the standard convention established by Damgård [11], and Bellare and Rogaway [7], these notions were formalized for hash function families, indexed by a (publicly known) key K. Current practical hash functions however do not have explicit keys. In fact, it is not even clear what the family is that they belong to, so it is rather contrived to regard SHA-256 as a randomly drawn member of such a family. Instead, the always-notions aSec and aPre capture the intuition that a hash function ought to be (second-)preimage resistant for *all* members of the family, so that it doesn't matter which one is actually used. Alternatively, one could see the aSec and aPre notions as the the natural extensions to (second-)preimage resistance of Rogaway's human-ignorance approach to collision-resistant hashing with unkeyed compression functions [27]. (See [2] for a subsequent work on property preservation for iterations of unkeyed compression

Table 1. Overview of constructions and the properties they preserve. Each row in the table represents a hash function construction, each column a security notion of [28]. The symbol "Y" means that the notion is provably preserved by the construction; "N" means that it is not preserved, in the sense that we come up with a counterexample; "?" means that neither proof nor counterexample are known. Underlined entries were known, all other results are new.

Scheme	Coll	Sec	aSec	eSec	Pre	aPre	ePre
Strengthened MD [22, 11]	$\underline{Y}$	N	N	$\underline{N}$	N	N	Y
Linear [7]	N	N	N	$\underline{N}$	N	N	Y
XOR-Linear [7]	Y	N	N	$\underline{Y}$	N	N	Y
Shoup's [30]	Y	N	N	$\underline{Y}$	N	N	Y
Prefix-free MD [9]	$\underline{N}$	N	N	N	N	N	Y
Randomized [13]	Y	N	N	N	N	N	Y
HAIFA [8]	$\underline{Y}$	N	N	N	N	N	Y
Enveloped MD [4]	$\underline{Y}$	N	N	N	N	N	Y
Strengthened Merkle Tree [20]	$\underline{Y}$	N	N	N	N	N	Y
Tree Hash [7]	N	N	N	N	N	N	Y
XOR Tree [7]	?	?	N	?	Y	N	Y
ROX	Y	Y	Y	Y	Y	Y	Y

functions.) In this sense, the aSec and aPre notions strengthen the standard notions of second-preimage resistance and preimage resistance, respectively, in the way needed to say that a fixed function such as SHA-256 is Sec and Pre secure. They therefore inherit the practical applications of Sec and Pre security, and are thus the right notions to consider when instantiating Cramer-Shoup encryption or Unix-like password storage with a fixed function like SHA-256. The formal definitions of all seven notions are recalled in Section 2.

Existing constructions. Let us now take a closer look at a number of existing constructions to see which of the seven notions of [28] they preserve. Our findings are summarized in Table 1, which we see as the main research contribution of our paper. Except for the few entries in the table with question marks, we come up with either proofs or counterexamples in support of our claims. We found for example that the ubiquitous SMD construction preserves Coll and ePre security, but surprisingly fails to preserve any of the other notions. Of the eleven schemes in the table, none preserves all seven notions. In fact, the best-performing constructions in terms of property preservation are the XOR Linear hash and Shoup's hash, which still preserve only three of the seven notions (Coll, eSec, and ePre). The XOR Tree hash is the only iteration to preserve Pre, and none of the schemes preserve Sec, aSec or aPre. Remember that the latter two are particularly relevant for the security of practical hash functions because they do not rely on the compression functions being chosen at random from a family.

Preserving all properties: the ROX Construction. This rather poor state of affairs may leave one wondering whether preserving all seven notions is possible at all. We answer this question in the affirmative, but, quite controversially, were only able to do so in the random oracle model. We explicitly do *not* model the compression function itself as a random oracle however. While we view the main interest of our construction to be a feasibility result for seven-property-preserving hashing, we do have reasons to believe that our construction makes very "reasonable" use of the random oracle. Allow us to explain.

Our Random-Oracle-XOR (ROX) construction draws largely on the XOR-linear hash [7] and Shoup's hash [30]. The latter is an extension of SMD where a logarithmic (in the message length) number of masks are XORed into the chaining value. We take the same approach, but have the masks generated by applying a random oracle to 170-bit inputs, for a security level of 80 bits. To hash an ℓ-block message, we query the random oracle on a number of domain points that is logarithmic in ℓ. This limited use of the random oracle has the important practical ramification that the function instantiating it need not be as efficient as the compression function, and can therefore be made with large security margins. We'll come back to candidate instantiations in Section 4.

The idea of generating the masks through a random oracle is not new; in fact, it was explicitly suggested at two separate occasions by Mironov [23, 24]. The idea was discarded in [23] for trivializing the problem, but was revisited in [24] as a viable way to obtain shorter keys for eSec-secure hashing. Indeed, if one assumes the existence of random oracles with very large domains, then one can simply use the random oracle to do the hashing. The ROX construction, on the other hand, still uses a real compression function in the chaining, and uses a small-domain random oracle to preserve all seven notions of [28] using a very short key, including the important aSec and aPre notions.[5] Moreover, we do so without changing the syntax of the compression function [8] or doubling its output size [19], both of which can come at a considerable performance penalty.

WHAT ABOUT OTHER PROPERTIES? The seven security notions formalized by [28] are certainly not the only ones that are of interest. Kelsey and Kohno [14] suggest chosen-target forced-prefix security, which can be seen as a special form of multi-collision resistance, as the right goal to stop Nostradamus attacks. Bellare and Ristenpart [4], following previous work by Coron et al. [9] and Bellare et al. [3], formalize pseudorandom oracle preservation (PRO-Pr) and pseudorandom function preservation (PRF-Pr) as goals. Their EMD construction is shown to be PRO-Pr, PRF-Pr and to preserve collision resistance. More recently, and independently of this work, Bellare and Ristenpart [5] study the Coll, eSec, PRO, PRF, and MAC (unforgeability) preservation of various iterations, including the SMD, Prefix-free MD, Shoup, and EMD iterations that we study. Their work does not cover the five other notions of [28], while our work does not cover the PRO, PRF, and MAC properties. We leave the study of the preservation of these properties by our $\mathcal{ROX}$ construction to future work.

---

[5] While ROX itself is an explicitly keyed construction, its preservation of aSec/aPre implies that the instantiating compression function need not be. Indeed, when instantiated with a fixed aSec/aPre-secure compression function like SHA-256, then the resulting iterated hash is aSec/aPre-secure and therefore also Sec/Pre-secure. ROX thereby provides a secure way of iterating unkeyed (second-)preimage resistant compression functions.

2 Security Definitions

In this section, we explain the security notions for hash functions of [28]. Let us begin by establishing some notation. Let $\mathbb{N} = \{0, 1, \ldots\}$ be the set of natural numbers and $\{0,1\}^*$ be the set of all bit strings. If $k \in \mathbb{N}$, then $\{0,1\}^k$ denotes the set of all k-bit strings and $\{0,1\}^{k\times *}$ denotes the set of all bit strings of length an integer multiple of k. The empty string is denoted ε. If b is a bit then $\overline{b}$ denotes its complement. If x is a string and $i \in \mathbb{N}$, then $x^{(i)}$ is the i-th bit of x and x^i is the concatenation of i copies of x. 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. We occasionally write $\langle k\rangle$ when the length is clear from the context. If S is a set, then $x \xleftarrow{\$} S$ denotes the uniformly random selection of an element from S. We let $y \leftarrow \mathsf{A}(x)$ and $y \xleftarrow{\$} \mathsf{A}(x)$ be the assignment to y of the output of a deterministic and randomized algorithm A, respectively, when run on input x.

An *adversary* is an algorithm, possibly with access to oracles. To avoid trivial lookup attacks, it will be our convention to include in the time complexity of an adversary A its running time and its code size (relative to some fixed model of computation).

Security Notions for Keyed Hash Functions. Formally, a *hash function family* is a function $\mathrm{H} : \mathcal{K} \times \mathcal{M} \to \mathcal{Y}$ where the key space $\mathcal{K}$ and the target space $\mathcal{Y}$ are finite sets of bit strings. The message space $\mathcal{M}$ could be infinitely large; we only assume that there exists at least one $\lambda \in \mathbb{N}$ such that $\{0,1\}^\lambda \subseteq \mathcal{M}$. We treat (fixed input length) compression functions and (variable input length) hash functions just the same, the former being simply a special case of the latter.

The seven security notions from [28] are the standard three of *collision resistance* (Coll), *preimage resistance* (Pre), and *second-preimage resistance* (Sec), and the *always*- and *everywhere*-variants of (second-)preimage resistance (aPre, aSec, ePre, and eSec). The advantage of an adversary A in breaking H under security notion atk is given by $\mathbf{Adv}^{\mathrm{atk}}_{\mathrm{H}}(\mathsf{A}) = \Pr[\mathrm{Exp}_{\mathrm{atk}} : M \neq M' \text{ and } \mathrm{H}(K, M) = \mathrm{H}(K, M')]$ if atk $\in \{\mathrm{Coll}, \mathrm{Sec}[\lambda], \mathrm{eSec}, \mathrm{aSec}[\lambda]\}$, and by $\mathbf{Adv}^{\mathrm{atk}}_{\mathrm{H}}(\mathsf{A}) = \Pr[\mathrm{Exp}_{\mathrm{atk}} : \mathrm{H}(K, M') = Y]$ if atk $\in \{\mathrm{Pre}[\lambda], \mathrm{ePre}, \mathrm{aPre}[\lambda]\}$, 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]	$(K, St) \xleftarrow{\$} \mathsf{A}\,;\ M \xleftarrow{\$} \{0,1\}^\lambda\,;\ Y \leftarrow \mathrm{H}(K, M)\,;\ M' \xleftarrow{\$} \mathsf{A}(Y, St)$

We say that A is (t, ϵ) atk-secure if no adversary running in time at most t has advantage more than ϵ. When giving results in the random oracle model, we will talk about $(t, q_{\mathrm{RO}}, \epsilon)$ atk-secure schemes, where q_{RO} is the total number of queries that A makes to its random oracles.

Note that the security notions above do not insist that the colliding message M' be of length λ. It is our conscious choice to focus on arbitrary-length security here, meaning that adversaries may find collisions between messages of varying lengths. In practice, the whole purpose of hash iterations is to extend the domain of a compression function to arbitrary lengths, so it makes perfect sense to require that the hash function withstands attacks using messages of different lengths.

3 Properties Preserved by Existing Constructions

In this section we take a closer look at eleven hash iterations that previously appeared in the literature, and check which of the seven security properties from [28] they preserve. The algorithms are described in Fig. 1, the results of our analysis are summarized in Table 1.

As mentioned in the previous section, we focus on arbitrary-length security in this paper. Allowing for arbitrary-length message attacks invariably seems to require some sort of message padding (unstrengthened MD does not preserve collision resistance), but care must be taken when deciding on the padding method: one method does not fit all. This was already observed by Bellare and Rogaway [7], who proposed an alternative form of strengthening where a final block containing the message length is appended and processed with a different key than the rest of the iteration. This works fine in theory, but since current compression functions are not keyed, it is not clear how this construction should be instantiated in practice. In absence of a practical generic solution, we chose to add standard one-zeroes padding and length strengthening to all chaining iterations that were originally proposed without strengthening. For tree iterations we use one-zeroes padding for the message input at the leaves, and at the root make one extra call to the compression function on input the accumulated hash value concatenated with the message length. (Standard length strengthening at the leaves fails to preserve even collision resistance here.) These strengthening methods sometimes help but never harm for property preservation.

Strengthened Merkle-Damgård. The *Strengthened Merkle-Damgård* ($\mathcal{SMD}$) construction is known to preserve collision resistance [11] and to not preserve eSec security [7]. In the following two theorems we prove that it also preserves ePre security, but does not preserve Sec, aSec, Pre, and aPre security. τ_{F} is the time required for an evaluation of F and $\ell = \lceil (\lambda + 2n)/b \rceil$ where $\lambda = |M|$.

Theorem 1. *If* F *is* (t', ϵ') ePre-*secure, then* $\mathcal{SMD}_{\mathrm{F}}$ *is* (t, ϵ) ePre-*secure for* $\epsilon = \epsilon'$ *and* $t = t' - \ell \cdot \tau_{\mathrm{F}}$.

Proof. Given an ePre-adversary A against $\mathcal{SMD}_{\mathrm{F}}$, consider the following ePre-adversary B against F. B runs A to obtain the target value Y and outputs the same string Y. When it gets a random key K it runs A on the same key to obtain

Algorithm $\mathcal{SMD}_{\mathrm{F}}(K, M)$: $m_1 \\| \ldots \\| m_\ell \leftarrow$ ls-pad(M) ; $h_0 \leftarrow IV$ For $i = 1 \ldots \ell$ do $h_i \leftarrow \mathrm{F}(K, m_i \\| h_{i-1})$ Return h_ℓ	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 $h_i \leftarrow \mathrm{F}(K_i, m_i \\| h_{i-1})$ 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 $h_i \leftarrow \mathrm{F}(K, m_i \\| (h_{i-1} \oplus K_{i-1}))$ 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 $h_i \leftarrow \mathrm{F}(K, m_i \\| (h_{i-1} \oplus K_{\nu(i)}))$ Return h_ℓ
Algorithm $\mathcal{PfMD}_{\mathrm{F}}(K, M)$: $m_1 \\| \ldots \\| m_\ell \leftarrow$ pf-pad(M) ; $h_0 \leftarrow IV$ For $i = 1, \ldots, \ell$ do $h_i \leftarrow \mathrm{F}(K, m_i \\| h_{i-1})$ Return h_ℓ	Algorithm $\mathcal{EMD}_{\mathrm{F}}(K, M)$: $m_1 \\| \ldots \\| m_\ell \leftarrow$ emd-pad(M) ; $h_0 \leftarrow IV_1$ For $i = 1 \ldots \ell - 1$ do $h_i \leftarrow \mathrm{F}(K, m_i \\| h_{i-1})$ Return $h_\ell \leftarrow \mathrm{F}(K, h_{\ell-1} \\| m_\ell \\| IV_2)$
Algorithm $\mathcal{HAIFA}_{\mathrm{F}}(K, M)$: $m_1 \\| \ldots \\| m_\ell \leftarrow$ oz-pad$(M, i \cdot b)$; $h_0 \leftarrow IV$ $ctr \leftarrow 0$; $S \xleftarrow{\$} \{0,1\}^s$ // S *is a salt* For $i = 1 \ldots \ell - 1$ do $ctr \leftarrow ctr + b$; $h_i \leftarrow \mathrm{F}(K, m_i \\| \langle ctr \rangle_l \\| S \\| h_{i-1})$ $h_\ell \leftarrow \mathrm{F}(K, m_\ell \\| \langle \lvert M \rvert \rangle \\| S \\| h_{\ell-1})$ Return S, h_ℓ	Algorithm $\mathcal{RH}_{\mathrm{F}}(K \\| R, M)$: $m_1 \\| \ldots \\| m_\ell \leftarrow$ sf-pad(M) $h_0 \leftarrow \mathrm{F}(K, R \\| IV)$ For $i = 1 \ldots \ell$ do $h_i \leftarrow \mathrm{F}(K, (m_i \oplus R) \\| h_{i-1})$ Return h_ℓ

Algorithm $\mathcal{SMT}_{\mathrm{F}}(K, M)$:
$m_1 \\| \ldots \\| m_\ell \leftarrow$ tpad(M)
For $j = 1, \ldots, a^{d-1}$ do
$h_{1,j} \leftarrow \mathrm{F}(K, m_{(j-1)a+1} \\| \ldots \\| m_{ja})$
For $i = 2, \ldots, d$ and $j = 1, \ldots, a^{d-i}$ do
$h_{i,j} \leftarrow \mathrm{F}(K, h_{i-1,(j-1)a+1} \\| \ldots \\| h_{i-1,ja})$
$h_{d+1,1} \leftarrow \mathrm{F}(K, h_{d,1} \\| \langle \lvert M \rvert \rangle_{n(a-1)})$
Return $h_{d+1,1}$
	Algorithm $\mathcal{TH}_{\mathrm{F}}(K_1 \\| \ldots \\| K_{d+1}\ ,\ M)$:
	$m_1 \\| \ldots \\| m_\ell \leftarrow$ tpad(M)
	For $j = 1, \ldots, a^{d-1}$ do
	$h_{1,j} \leftarrow \mathrm{F}(K_1, m_{(j-1)a+1} \\| \ldots \\| m_{ja})$
	For $i = 2, \ldots, d$ and $j = 1, \ldots, a^{d-i}$ do
	$h_{i,j} \leftarrow \mathrm{F}(K_i, h_{i-1,(j-1)a+1} \\| \ldots \\| h_{i-1,ja})$
	$h_{d+1,1} \leftarrow \mathrm{F}(K_{d+1}, h_{d,1} \\| \langle \lvert M \rvert \rangle_{n(a-1)})$
	Return $h_{d+1,1}$

Algorithm $\mathcal{XTH}_{\mathrm{F}}(K \\| K_1 \\| \ldots \\| K_{d+1}\ ,\ M)$:
$m_1 \\| \ldots \\| m_\ell \leftarrow$ tpad(M)
For $j = 1, \ldots, a^{d-1}$ do
$h_{1,j} \leftarrow \mathrm{F}(K, (m_{(j-1)a+1} \\| \ldots \\| m_{ja}) \oplus K_1)$
For $i = 2, \ldots, d$ and $j = 1, \ldots, a^{d-i}$ do
$h_{i,j} \leftarrow \mathrm{F}(K, (h_{i-1,(j-1)a+1} \\| \ldots \\| h_{i-1,ja}) \oplus K_i)$
$h_{d+1,1} \leftarrow \mathrm{F}(K, (h_{d,1} \\| \langle \lvert M \rvert \rangle_{n(a-1)}) \oplus K_{d+1})$
Return $h_{d+1,1}$
	Padding algorithms:
	oz-pad$(M, x) = M \\| 10 0^{x - \lvert M \rvert - 2}$
	ls-pad$(M) =$ oz-pad$(M, x) \\| \langle \lvert M \rvert \rangle_b$
	where $x = \lceil (\lvert M \rvert + 2)/b \rceil \cdot b$
	emd-pad$(M) =$ oz-pad$(M, x) \\| \langle \lvert M \rvert \rangle_{64}$
	where $x = \lceil (\lvert M \rvert + 66)/b \rceil \cdot b - 64$
	tpad$(M) =$ oz-pad(M, x)
	where $x = a^{\lceil \log_a \lvert M \rvert \rceil} \cdot n$

Fig. 1. Some existing iterative hash constructions. Chaining iterations $\mathcal{SMD}$, $\mathcal{LH}$, $\mathcal{XLH}$, $\mathcal{SH}$, $\mathcal{PfMD}$, $\mathcal{RH}$, and $\mathcal{EMD}$ use a compression function $\mathrm{F} : \{0,1\}^k \times \{0,1\}^{b+n} \to \{0,1\}^n$; $\mathcal{HAIFA}$ uses a compression function $\mathrm{F} : \{0,1\}^k \times \{0,1\}^{b+l+s+n} \to \{0,1\}^n$. Tree iterations $\mathcal{SMT}$, $\mathcal{TH}$, and $\mathcal{XTH}$ use a compression function $\mathrm{F} : \{0,1\}^k \times \{0,1\}^{an} \to \{0,1\}^n$. Strings $IV, IV_1, IV_2 \in \{0,1\}^n$ are fixed initialization vectors. Padding algorithms are given on the bottom right; pf-pad(M) and sf-pad(M) are any prefix-free padding and suffix-free padding algorithms, respectively. The function $\nu(i)$ is the largest integer j such that $2^j | i$.

a preimage message M'. Let $m'_1 \| \ldots \| m'_\ell \leftarrow$ ls-pad(M') and let $h'_{\ell-1}$ be the one-but-last chaining value computed in an execution of $\mathcal{SMD}_{\mathrm{F}}(K, M')$. Algorithm B outputs $m'_\ell \| h'_{\ell-1}$ as its own preimage.

While at first sight the above proof may seem to go through for Pre and aPre security as well, this is not the case. The target point Y in a Pre attack on F is distributed as $\mathrm{F}(K, m \| h)$ for a random $m \| h \xleftarrow{\$} \{0,1\}^{b+n}$. But the target point for the iterated structure $\mathcal{SMD}_{\mathrm{F}}$ is generated as $\mathcal{SMD}_{\mathrm{F}}(K, M)$ for a random $M \xleftarrow{\$} \{0,1\}^\lambda$. These two distributions can actually be very different, as is illustrated by the following counterexample.

Theorem 2. *For* $\mathrm{atk} \in \{\mathrm{Sec}, \mathrm{aSec}, \mathrm{Pre}, \mathrm{aPre}\}$*, if there exists a* (t, ϵ) atk*-secure compression function* $\mathrm{G} : \mathcal{K} \times \{0,1\}^{b+n} \to \{0,1\}^{n-1}$*, then there exists a* $(t, \epsilon - 1/2^n)$ atk*-secure compression function* $\mathrm{CE}_1 : \mathcal{K} \times \{0,1\}^{b+n} \to \{0,1\}^n$ *and an adversary* A *running in one time step with* $\mathrm{atk}[\lambda]$*-advantage one in breaking* $\mathcal{SMD}_{\mathrm{CE}_1}$*.*

Proof. For any compression function G, consider CE_1 given by

$$\begin{aligned} \mathrm{CE}_1(K, m\|h) &= IV && \text{if } h = IV \\ &= \mathrm{G}(K, m\|h) \parallel \overline{IV}^{(n)} && \text{otherwise .} \end{aligned}$$

If G is (t, ϵ) atk secure, then CE_1 is $(t, \epsilon - 1/2^n)$ atk secure; we refer to the full version [1] for the proof. From the construction of CE_1, it is clear that $\mathcal{SMD}_{\mathrm{CE}_1}(K, M) = IV$ for all $M \in \{0,1\}^*$. Hence, the adversary can output any message M' as its (second) preimage.

Linear hash. The *Linear Hash* ($\mathcal{LH}$) [7] uses ℓ different keys for ℓ-block messages, because it calls the compression function on a different key at every iteration. The Linear Hash is known to preserve eSec-security for same-length messages, but Bellare and Rogaway claim [7] that length-strengthening does not suffice to preserve eSec for different-length messages. The following theorem confirms their claim, and also shows that $\mathcal{LH}$ does not preserve Coll. The counterexample CE_1 of Theorem 2 can be used to disprove the preservation of Sec, aSec, Pre and aPre-security. A proof similar to that of Theorem 1 can be used to show that $\mathcal{LH}$ does preserve ePre-security.

Theorem 3. *For any* $\mathrm{atk} \in \{\mathrm{Coll}, \mathrm{eSec}\}$*, if there exists a* (t, ϵ) atk*-secure compression function* $\mathrm{G} : \{0,1\}^k \times \{0,1\}^{b+n} \to \{0,1\}^{n-2}$*, then there exists a* (t, ϵ) atk*-secure compression function* $\mathrm{CE}_2 : \{0,1\}^k \times \{0,1\}^{b+n} \to \{0,1\}^n$ *and an adversary* A *running in one step time with* atk*-advantage* $1/4$ *in breaking* $\mathcal{LH}_{\mathrm{CE}_2}$*.*

Proof. For any compression function G, consider CE_2 given by

$$\begin{aligned} \mathrm{CE}_2(K, m\|h) &= IV && \text{if } m\|h = 010^{b-2}\|IV \\ &= 0^{n-1} \parallel \overline{IV}^{(n)} && \text{if } (K^{(1)} = 0 \text{ and } m\|h = \langle 1\rangle_b\|IV) \\ & && \text{or } (K^{(1)} = 1 \text{ and } m\|h = \langle b+1\rangle_b\|IV) \\ &= \mathrm{G}(K, m\|h) \parallel 1 \parallel \overline{IV}^{(n)} && \text{otherwise ,} \end{aligned}$$

In the full version [1] we prove that if G is (t, ϵ) atk-secure for $\mathrm{atk} \in \{\mathrm{Coll}, \mathrm{eSec}\}$, then CE_2 is (t, ϵ) atk-secure. When iterating CE_2 through $\mathcal{LH}_{\mathrm{CE}_2}$ with independent keys $K_1\|K_2\|K_3$, one can easily see that if $K_2^{(1)} = 0$ and $K_3^{(1)} = 1$, then messsages $M = 0$ and $M' = 010^{b-1}$ both hash to $0^{n-1}\|\overline{IV}^{(n)}$. Since in the Coll and eSec games this case happens with probability 1/4, we have attacks satisfying the claim in the theorem.

XOR-Linear Hash. The *XOR-Linear Hash* ($\mathcal{XLH}$) [7] uses keys that consist of a compression function key K and ℓ masking keys $K_1, \ldots, K_\ell \in \{0,1\}^n$. It is known to preserve eSec security [7]. It can also be seen to preserve Coll and ePre by similar arguments as used for $\mathcal{SMD}$ and $\mathcal{LH}$. Counterexample CE_1 can be used to show that aSec and aPre are not preserved: the adversary gets to choose the key in these notions, so it can choose $K_1 = \ldots = K_\ell = 0^n$ so that $\mathcal{XLH}$ boils down to $\mathcal{SMD}$. In the following we show that the $\mathcal{XLH}$ construction does not preserve Sec or Pre security either.

Theorem 4. *For any* $\mathrm{atk} \in \{\mathrm{Sec}, \mathrm{Pre}\}$*, if there exists a* (t, ϵ) atk-*secure compression function* $\mathrm{G} : \mathcal{K} \times \{0,1\}^{b+n} \to \{0,1\}^{n-1}$*, then there exists a* $(t, \epsilon + 1/2^b)$ atk-*secure compression function* $\mathrm{CE}_3 : \mathcal{K} \times \{0,1\}^{b+n} \to \{0,1\}^n$ *and an adversary* A *running in one step time with* $\mathrm{atk}[\lambda]$*-advantage one in breaking* $\mathcal{XLH}_{\mathrm{CE}_3}$.

Proof. For any $\lambda \leq 2^b$ and compression function G, consider CE_3 given by

$$\begin{aligned} \mathrm{CE}_3(K, m\|h) &= 0^n && \text{if } m = \langle\lambda\rangle_b \\ &= \mathrm{G}(K, m\|h)\|1 && \text{otherwise} . \end{aligned}$$

In the full version [1] we prove that if G is (t, ϵ) Sec or Pre-secure, then CE_3 is $(t, \epsilon + 1/2^b)$ Sec or Pre-secure. It is easy to see that, when iterated through $\mathcal{XLH}_{\mathrm{CE}_3}$, the hash of any λ-bit message is 0^n. A $\mathrm{Pre}[\lambda]$ adversary can therefore simply output any $M' \in \{0,1\}^\lambda$, a $\mathrm{Sec}[\lambda]$ adversary can output any $M' \neq M \in \{0,1\}^\lambda$.

Shoup's Hash. The iteration due to Shoup ($\mathcal{SH}$) [30] is similar to the XOR-Linear hash but uses a different key scheduling that reduces the key length to logarithmic in the message length, rather than linear. Shoup's hash is known to preserve eSec-security [30], and it can be shown to preserve Coll and ePre-security as well. The proofs are very similar to the case of SMD, and hence omitted. Counterexample CE_1 disproves preservation of aSec and aPre-security, and counterexample CE_3 disproves preservation of Sec and Pre.

Prefix-free Merkle-Damgård. Bellare and Ristenpart showed [4] that the *prefix-free Merkle-Damgård* construction ($\mathcal{PfMD}$) [9] does not preserve Coll security. The counterexample of [7] can also be used to show that it does not preserve eSec, and counterexample CE_1 can be used to disprove the preservation of Sec, aSec, Pre and aPre. Finally, using a proof similar to that for $\mathcal{SMD}$, one can show that ePre-security is preserved.

Another variant of $\mathcal{PfMD}$ by [9] prepends the message length encoding to the message in advance. The security results of this scheme easily follow from the ones for the $\mathcal{SMD}$ construction.

Randomized hash. The *Randomized Hash* ($\mathcal{RH}$) [13] XORs each message block with a random value $R \in \{0,1\}^b$. The construction was originally proved to be eSec secure by making stronger assumptions on the underlying compression function. Its pure security preservation characteristics (i.e., assuming only the eSec security of the compression function) were never studied. In our security analysis of $\mathcal{RH}$ treating the value R as either randomness per message or fixed long term key yields identical results with respect to seven property preservation.

By arguments similar to the case of $\mathcal{SMD}$, one can show that $\mathcal{RH}$ preserves Coll and ePre security, but none of the other notions are preserved. Counterexample CE_1 can be used to contradict preservation of Sec, aSec, Pre, and ePre, and the counterexample of [7] can be used to contradict preservation of eSec.

HAIFA. While the newly proposed *HAsh Iterative FrAmework* ($\mathcal{HAIFA}$) [8] does preclude a number of specific attacks [12, 15, 14] to which $\mathcal{SMD}$ admits, they perform exactly the same in terms of preservation of our security notions. Similar proofs as for $\mathcal{SMD}$ can be used to show that $\mathcal{HAIFA}$ preserves Coll and ePre-security, counterexample CE_1 can be used to contradict the preservation of Sec, aSec, Pre, and aPre, and the counterexample of [7] applies to contradict preservation of eSec.

Enveloped Merkle-Damgård. The *enveloped Merkle-Damgård* ($\mathcal{EMD}$) construction [4] is known to preserve collision resistance, pseudo-random-oracle, and pseudo-random function behavior. For the seven security notions that we consider, however, it does not perform better than $\mathcal{SMD}$. Counterexample CE_1 of Theorem 2 can be used (setting $IV = IV_2$) to show that neither of Sec, aSec, Pre, or aPre are preserved. An adaptation of the counterexample of [7] shows that eSec is not preserved either. Preservation of ePre on the other hand can be proved in a similar way as done in Theorem 1.

Strengthened Merkle Tree. We consider here the strengthened Merkle tree [20], the Tree Hash [7], and the XOR Tree Hash [7]. For conciseness we do not cover other tree iterations that have appeared in the literature (e.g. [17, 29]). The Merkle tree [20] in its most basic form (i.e., without length strengthening) suffers from a similar anomaly as basic Merkle-Damgård in that it does not preserve Coll for arbitrary-length messages. We therefore consider the strengthened variant $\mathcal{SMT}$ here, depicted in Fig. 1. We believe $\mathcal{SMT}$ is commonly known to preserve Coll, but we reprove this in the full version [2] for completeness. The notion of ePre is easily seen to be preserved as well. It can be seen not to preserve eSec by a counterexample similar to that of [7] given in the full version [2]. $\mathcal{SMT}$ also fails to preserve Sec, aSec, Pre, and aPre however, as shown in the following theorem.

Theorem 5. *For any* $\mathrm{atk} \in \{\mathrm{Sec}, \mathrm{aSec}, \mathrm{Pre}, \mathrm{aPre}\}$*, if there exists a* (t', ϵ') atk-*secure compression function* $\mathrm{G} : \mathcal{K} \times \{0,1\}^{an} \to \{0,1\}^{n-2}$*, then there exists a* (t, ϵ) atk-*secure compression function* $\mathrm{CE}_4 : \mathcal{K} \times \{0,1\}^{an} \to \{0,1\}^n$ *for* $\epsilon = \epsilon' + 1/2^{n-1}$*,* $t = t'$*, and an adversary* A *running in one step time with* $\mathrm{atk}[\lambda]$ *advantage* 1 *in breaking* $\mathcal{SMT}_{\mathrm{CE}_4}$*.*

Proof. For any compression function G, consider CE_4 given by

$$\begin{aligned}\mathrm{CE}_4(K, m_1\|\ldots\|m_a) &= 0^n && \text{if } m_a = 0^n \\ &= 1^n && \text{if } m_{a-1} = 0^n \text{ and } m_a \neq 0^n \\ &= \mathrm{G}(K, m_1\|\ldots\|m_a) \,\|\, 10 && \text{otherwise .}\end{aligned}$$

We prove in the full version [1] that the bounds mentioned above hold for the atk security of CE_4. It is easy to see that, due to the one-zeroes padding to a^d

bits, any message of length $a^{d-1}-1 \leq \lambda \leq a^d-1$ hashes to 1^n, leading to trivial constant-time attacks for any such length λ.

Tree Hash. The unstrengthened Tree Hash ($\mathcal{TH}$) was proposed in [7] for same-length messages; we consider the strengthened variant here. It is a variant of $\mathcal{SMT}$ where at each level i of the tree the compression functions use an independent key K_i. It can be seen to preserve ePre for the same reasons as the $\mathcal{SMT}$ construction. Our counterexample CE_4 can be used to exhibit the non-preservation of Sec, aSec, Pre and aPre security. The case of Coll and eSec are a bit more subtle, but the counterexample below shows that $\mathcal{TH}$ does not preserve these either.

Theorem 6. *For any* atk $\in \{\mathrm{Coll}, \mathrm{eSec}\}$*, if there exists a* (t', ϵ') atk-*secure compression function* $\mathrm{G} : \{0,1\}^k \times \{0,1\}^{an} \to \{0,1\}^{n-1}$*, then there exists a* (t, ϵ) atk-*secure compression function* $\mathrm{CE}_5 : \{0,1\}^k \times \{0,1\}^{an} \to \{0,1\}^n$ *for* $\epsilon = \epsilon'$*,* $t = t'$*, such that there exists an* eSec-*adversary breaking the eSec security of* $\mathcal{TH}_{\mathrm{CE}_5}$ *in constant time with advantage* $1/4$*.*

Proof. For any compression function G, consider CE_5 given by

$$\begin{aligned}\mathrm{CE}_5(K, M) &= 10^{n-1} && \text{if } M = (10^{n-1})^a\\ &= 1^n && \text{if } \big(K^{(1)} = 0 \text{ and } M = (10^{n-1})^{a-1} \| \langle (a-1)n \rangle_n\big)\\ & && \text{or } \big(K^{(1)} = 1 \text{ and } M = (10^{n-1})^{a-1} \| \langle (a^2-1)n \rangle_n\big)\\ &= 0 \,\|\, \mathrm{G}(K, M) && \text{otherwise}\ .\end{aligned} \tag{1}$$

We prove in the full version [2] that CE_5 is (t, ϵ) atk-secure whenever G is (t, ϵ) atk-secure, for atk $\in \{\mathrm{Coll}, \mathrm{eSec}\}$.

Let $M = (10^{n-1})^{a-1}$ and $M' = (10^{n-1})^{a^2-1}$. Note that $\mathrm{tpad}(M) = (10^{n-1})^a$ and $\mathrm{tpad}(M') = (10^{n-1})^{a^2}$, where tpad is the tree padding algorithm of Fig. 1. If $\mathcal{TH}_{\mathrm{CE}_5}$ is instantiated with keys $K_1 \| K_2 \| K_3$ such that $K_2^{(1)} = 0$ and $K_3^{(1)} = 1$, then one can verify that $\mathcal{TH}_{\mathrm{CE}_5}(K_1 \| K_2 \| K_3, M') = \mathcal{TH}_{\mathrm{CE}_5}(K_1 \| K_2 \| K_3, M) = 1^n$. Hence, the adversary that outputs M and M' as colliding message pair has advantage $1/4$ in winning the Coll and eSec games.

XOR Tree. The unstrengthened XOR Tree ($\mathcal{XTH}$) was proposed in [7] for fixed-length messages; we consider the strengthened variant here. It is again a variant of the Merkle tree, where the inputs to the compression functions on level i are XORed with a key $K_i \in \{0,1\}^{an}$. As for all other iterations, it is straightforward to see that $\mathcal{XTH}$ preserves ePre; we omit the proof. Quite remarkably, the masking of the entire input to the compression function makes it the only iteration in the literature that preserves Pre, while at the same time it seems to stand in the way of even proving preservation of Coll. It does not preserve aSec or aPre because the adversary can choose $K_i = 0^{an}$ and apply counterexample CE_4. We were unable to come up with either proof or counterexample for Coll, Sec, and eSec, leaving these as an open question. The proof of preservation of Pre is given in the full version [1].

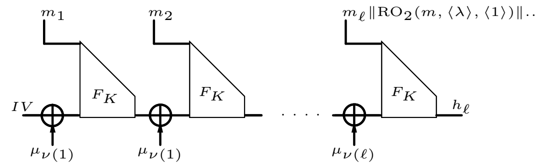

Fig. 2. The ROX Construction. The message is padded with bits generated by $\mathrm{RO}_2(K, \mathfrak{m}, \langle\lambda\rangle, \langle i\rangle)$, where $\mathfrak{m}$ are the first k bits of M. The last block must contain at least $2n$ padding bits, otherwise an extra padding block is added. In the picture above, IV is the initialization vector, $\nu(i)$ is the largest integer j such that $2^j | i$, and the masks $\mu_i \leftarrow \mathrm{RO}_1(K, \mathfrak{m}, \langle i\rangle)$.

4 The ROX Construction

We are now ready to present in detail our Random-Oracle-XOR (ROX) construction. Let $\mathrm{F} : \{0,1\}^k \times \{0,1\}^{b+n} \to \{0,1\}^n$ be a fixed-length compression function. Let 2^l be the maximum message length in bits; typically one would use $k = 80$ and $l = 64$. The construction uses two random oracles $\mathrm{RO}_1 : \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^{\lceil \log l \rceil} \to \{0,1\}^n$ and $\mathrm{RO}_2 : \{0,1\}^k \times \{0,1\}^l \times \{0,1\}^{\lceil \log b \rceil} \to \{0,1\}^{2n}$. These random oracles can be built from a single one by adding an extra bit to the input that distinguishes calls to RO_1 and RO_2. Our construction can be thought of as a variant of Shoup's hash, but with the masks being generated by RO_1 and the padding being generated by RO_2. More precisely, on input a message M, our padding function rox-pad outputs a sequence of b-bit message blocks

$$m_1 \| \ldots \| m_\ell \;=\; M \;\|\; \mathrm{RO}_2(\mathfrak{m}, \langle\lambda\rangle, \langle 1\rangle) \;\|\; \mathrm{RO}_2(\mathfrak{m}, \langle\lambda\rangle, \langle 2\rangle) \;\|\; \ldots\;,$$

where $\mathfrak{m}$ are the first k bits of M and $\lambda = |M|$. The padding adds a number of bits generated by RO_2 such that the final block m_ℓ contains at least $2n$ bits generated by RO_2, possibly resulting in an extra block consisting solely of padding. It is worth noting though that we do not have a separate length strengthening block. We assume that $\lambda \geq k$ because aPre security, and therefore seven-property-preservation as a whole, do not make sense for short messages. Indeed, the adversary can always exhaustively try the entire message space. To hash shorter messages, one should add a random salt to the message.

Let $\nu(i)$ be the largest integer j such that 2^j divides i, let $IV \in \{0,1\}^n$ be an initialization vector, and let $\mathfrak{m}$ be the first k bits of the message M. Our construction is described in pseudocode below; a graphical representation is given in Fig. 2.

Algorithm $\mathcal{ROX}_{\mathrm{F}}^{\mathrm{RO}_1,\mathrm{RO}_2}(K, M)$:
  $m_1 \| \ldots \| m_\ell \leftarrow \text{rox-pad}^{\mathrm{RO}_2}(M)$; $h_0 \leftarrow IV$
  For $i = 0, \ldots, \lfloor \log_2(\ell) \rfloor$ do $\mu_i \leftarrow \mathrm{RO}_1(K, \mathfrak{m}, \langle i\rangle)$
  For $i = 1 \ldots \ell$ do $g_i \leftarrow h_{i-1} \oplus \mu_{\nu(i)}$; $h_i \leftarrow \mathrm{F}(K, m_i \| g_i)$
  Return h_ℓ .

We want to stress that that the ROX construction does not require that the compression function accept an additional input that might be influenced by the attacker (such as a salt or a counter). We see this as an important advantage, since imposing additional requirements on the compression function may make compression functions even harder to design or less efficient.

It is quite standard in cryptography for new primitives to first find instantiations in the random oracle model, only much later to be replaced with constructions in the standard model. It is interesting to see how the random oracles in the ROX construction can be instantiated if one were to implement it in practice. For an 80-bit security level, our results suggest that we should take $k = 80$ and $n = 160$. This means that we need a random oracle that reduces about 170 bits to 160 bits. A first suggestion might be to re-use the compression function with, say, three times as many rounds as normal, and with some different values of the constants. This approach violates good cryptographic hygiene, however, by having the design of the random oracle depend on that of the surrounding scheme. Perhaps a better solution would be to use one or more calls to a blockcipher like AES that was designed independently of the compression function.

5 Properties Preserved by the ROX Construction

The following theorem states that the ROX construction preserves all seven security properties that we consider here. We give a proof sketch for the preservation of Coll and a full proof for aSec below; the other proofs can be found in the full version [2]. We only note that the proofs for Sec, aSec and eSec are in the programmable random oracle model [26]; that for the case of Pre and aPre non-programmable random oracles suffice; and that Coll and ePre are preserved in the standard model.

Theorem 7. *For* atk $\in \{\mathrm{Coll}, \mathrm{Sec}, \mathrm{eSec}, \mathrm{aSec}, \mathrm{Pre}, \mathrm{ePre}, \mathrm{aPre}\}$*, if the compression function* $\mathrm{F} : \{0,1\}^k \times \{0,1\}^{b+n} \to \{0,1\}^n$ *is* (t', ϵ') atk*-secure, then the iterated function* $\mathcal{ROX}_{\mathrm{F}}$ *is* $(t, q_{\mathrm{RO}}, \epsilon)$ atk*-secure in the random oracle for*

$$\begin{array}{llllll}
\epsilon = \epsilon' + \dfrac{q_{\mathrm{RO}}^2}{2^{2n}}, & t = t' - 2\ell \cdot \tau_{\mathrm{F}} & \textit{for } \mathrm{atk} = \mathrm{Coll} & (2)\\
\epsilon = \ell \cdot \epsilon' + \dfrac{q_{\mathrm{RO}}^2}{2^{2n}}, & t = t' - 2\ell \cdot \tau_{\mathrm{F}} & \textit{for } \mathrm{atk} = \mathrm{Sec} & (3)\\
\epsilon = \ell \cdot \epsilon' + \dfrac{q_{\mathrm{RO}}}{2^k} + \dfrac{q_{\mathrm{RO}}^2}{2^{2n}}, & t = t' - 2\ell \cdot \tau_{\mathrm{F}} & \textit{for } \mathrm{atk} = \mathrm{eSec} & (4)\\
\epsilon = \ell \cdot \epsilon' + \dfrac{q_{\mathrm{RO}}}{2^k} + \dfrac{q_{\mathrm{RO}}^2}{2^{2n}}, & t = t' - 2\ell \cdot \tau_{\mathrm{F}} & \textit{for } \mathrm{atk} = \mathrm{aSec} & (5)\\
\epsilon = \epsilon', & t = t' - \ell \cdot \tau_{\mathrm{F}} & \textit{for } \mathrm{atk} \in \{\mathrm{Pre}, \mathrm{ePre}\} & (6)\\
\epsilon = \epsilon' + \dfrac{q_{\mathrm{RO}}}{2^k}, & t = t' - \ell \cdot \tau_{\mathrm{F}} & \textit{for } \mathrm{atk} = \mathrm{aPre} & (7)
\end{array}$$

Here, τ_{F} *is the time required for an evaluation of* F *and* $\ell = \lceil(\lambda + 2n)/b\rceil$ *where* $\lambda = |M|$.

We repeat that above we do not model the compression function as a random oracle, but it is worth considering what the equations tell us if we do.

Assuming for simplicity that $\tau_{\mathrm{F}} = 1$, we know that a collision adversary running in $t' = 2^{n/2}$ steps has probability about 1/2 to find collisions in F, due to the birthday paradox, but only has probability $\epsilon' = 2^{-n/2}$ to find preimages or second preimages. Nevertheless, existing iterations cannot guarantee (second) preimage resistance against $2^{n/2}$-time adversaries, because they merely inherit their (second) preimage resistance by implication from collision resistance.[6] The ROX construction, on the other hand, can. Assuming that queries to $\mathrm{RO}_1, \mathrm{RO}_2$ take unit time and taking $k = n$, Equations (2), (3), (6) imply that an adversary running in time $t = 2^{n/2} - 2\ell \approx 2^{n/2}$ steps has probability at most $\epsilon = \ell \cdot 2^{-n/2} + 2^{n/2-k} + 2^{-n} \approx (\ell+1) \cdot 2^{-n/2}$ to find second preimages, and has probability at most $\epsilon' = 2^{-n/2} + 2^{n/2-n} \approx 2^{-n/2+1}$ to find preimages.

Proof (Equation (2) (Sketch)). If M, M' is a pair of colliding messages, then consider the two chains of compression function calls in the computation of $\mathcal{ROX}_{\mathrm{F}}(K, M) = \mathcal{ROX}_{\mathrm{F}}(K, M')$. If the inputs to the final call to F are different for M and M', then these inputs form a collision on F and we're done. If they are the same, then remember that at least $2n$ bits of these inputs are the output of $\mathrm{RO}_2(\mathfrak{m}, \langle\lambda\rangle, \langle i\rangle)$ and $\mathrm{RO}_2(\mathfrak{m}', \langle\lambda'\rangle, \langle j\rangle)$, respectively. If these are different queries to RO_2, yet their outputs are the same, then the adversary must have found a collision on RO_2; the odds of it doing so are bounded by $q_{\mathrm{RO}}^2/2^{2n}$. If these queries are the same, however, then we have that $\mathfrak{m} = \mathfrak{m}'$ and $\lambda = \lambda'$, and therefore that the masks in both chains $\mu_i = \mu_i' = \mathrm{RO}_1(K, \mathfrak{m}, \langle i\rangle)$. Identical chaining inputs to ℓ-th call to F must therefore be caused by identical outputs of the $(\ell-1)$-st call to F. If the inputs to the $(\ell-1)$-st call are different then we have a collision on F here, otherwise we repeat the argument to the $(\ell-2)$-nd call, and so on. A collision on F will be found unless $M = M'$. We refer to the full version [2] for a more detailed proof.

Proof (Equation (5)). Given an $\mathrm{aSec}[\lambda]$ adversary A against $\mathcal{ROX}_{\mathrm{F}}$ for any $\lambda \in \mathbb{N}$, we will construct an aSec adversary B against F. The overall strategy will be that B "embeds" his own challenge message at a random point in the chain, and hopes that A's output yields a second preimage at exactly the point in the chain where B has embedded his challenge.

Algorithm B runs A to obtain a key $K \in \{0,1\}^k$, responding to its random oracle queries by maintaining associative arrays $T_1[\cdot], T_2[\cdot]$. B outputs the same key K and is then given as input a random challenge message $m\|g \in \{0,1\}^{b+n}$. It chooses a random index $i^* \xleftarrow{\$} \{1, \ldots, \ell = \lceil(\lambda + 2n)/b\rceil\}$. We first explain how B can construct a message M of length λ so that $m_{i^*} = m$ in $m_1\| \ldots \|m_\ell \leftarrow$ rox-pad$^{\mathrm{RO}_2}(M)$; the rest of the message blocks are randomly generated. After that, we will show how g can be embedded into the chain such that $g_{i^*} = g$. If $i^* = 1$ then B sets $\mathfrak{m}$ to the first k bits of m, otherwise it chooses $\mathfrak{m} \xleftarrow{\$} \{0,1\}^k$

[6] For the Prefix-free MD [9] and EMD [4] iterations this is a bit paradoxical, because they were designed to preserve "random oracle behavior". Surely, (second) preimage resistance should fall under any reasonable definition of "random oracle behavior"? The caveat here is that the proof [4, Theorem 5.2] bounds the distinguishing probability to $O(q_{\mathrm{RO}}^2/2^n)$, so that the theorem statement becomes moot for $q_{\mathrm{RO}} = 2^{n/2}$.

and sets the first k bits of M to $\mathfrak{m}$. We distinguish between Type-I message blocks that only contain bits of M, Type-II message blocks of which the first $\lambda_b = (\lambda \bmod b)$ bits are the last λ_b bits of M and the remaining bits are generated by RO_2, and Type-III message blocks that consist entirely of bits generated by RO_2. Embedding m in a Type-I message block can simply be done by setting b bits of M to m starting at bit position $(i^* - 1)b + 1$. To embed m in a Type-II message block, B sets the last λ_b bits of M to the first λ_b bits of m, and programs the first $(b - \lambda_b)$ bits of $T_2[\mathfrak{m}, \langle\lambda\rangle, \langle 1\rangle] \parallel T_2[\mathfrak{m}, \langle\lambda\rangle, \langle 2\rangle] \parallel \ldots$ to be the last $(b - \lambda_b)$ bits of $\mathfrak{m}$. For Type-III blocks, B chooses M completely at random and sets b bits of $T_2[\mathfrak{m}, \langle\lambda\rangle, \langle 1\rangle] \parallel T_2[\mathfrak{m}, \langle\lambda\rangle, \langle 2\rangle] \parallel \ldots$ to m, starting at the $(b - \lambda_b + 1)$-st bit position. Bits of M and $T_2[\mathfrak{m}, \cdot]$ that are still undefined are chosen at random. If any of these table entries were already defined during A's first run, then B aborts. Notice however that A's view during the first run is independent of $\mathfrak{m}$, so its probability of making such a query is at most $q_{\mathrm{RO}}/2^k$.

To enforce that $g_{i^*} = g$ in the computation of $\mathcal{ROX}_{\mathrm{F}}^{\mathrm{RO}_1,\mathrm{RO}_2}(K, M)$, algorithm B runs the reconstruction algorithm of [30, 23] that, given message blocks $m_1, \ldots, m_{i^*}$ and chaining value g_{i^*}, outputs random mask values $\mu_0, \ldots, \mu_t$ such that the chaining input to the i^*-th compression function call is g_{i^*}. B's goal is to program these masks into RO_1 by setting $T_1[K, \mathfrak{m}, \langle i\rangle] \leftarrow \mu_i$ for $0 \le i \le t$, such that it is possible to check that the value for g_{i^*} obtained during the hash computation is indeed g. However, if any of the hash table entries $T_1[K, \mathfrak{m}, \langle i\rangle]$ for $0 \le i \le t$ has already been defined, then B aborts. This can only occur when A asked a query $\mathrm{RO}_1(K, \mathfrak{m}, \langle i\rangle)$ during its first phase, but again, the probability of it doing so is at most $q_{\mathrm{RO}}/2^k$ because its view is independent of $\mathfrak{m}$.

Algorithm B then runs A again on input target message M, responding to its random oracle queries as before, until it outputs a second preimage M'. Let $m'_0 \| \ldots \| m'_{\ell'} \leftarrow \text{rox-pad}^{\mathrm{RO}_2}(M')$ be the parsed messages. For the same arguments as in the proof of Equation (2) above, there must exist an index $I > 0$ such that $h_I = h'_I$ but $m_I \| g_I \neq m'_I \| g'_I$, unless A found a collision in the random oracle RO_2. If $i^* = I$, then B outputs $m'_I \| g'_I$.

B wins the game whenever A does and $i^* = I$, unless A succeeded in causing a collision in RO_2 or any of the values that are programmed in $\mathrm{RO}_1, \mathrm{RO}_2$ were already queried. Let E_1 be the event that at least one of the preprogrammed values is queried by A on a different input and E_2 be the event that A manages to find at least one collision in RO_2. Let ABORT be the event that B aborts, then

$$\Pr[\,\textsc{abort}\,] = \Pr[\,\mathrm{E}_1\,] + \Pr\left[\,\mathrm{E}_2 : \overline{\mathrm{E}_1}\,\right] \le \Pr[\,\mathrm{E}_1\,] + \Pr[\,\mathrm{E}_2\,]\,.$$

Since B perfectly simulates A's environment, the advantage of B is given by

$$\begin{aligned}\epsilon' &\ge \Pr[\,\mathsf{A} \text{ wins} \wedge i^* = I : \overline{\textsc{abort}}\,] \cdot \Pr[\,\overline{\textsc{abort}}\,] \\ &\ge \frac{\epsilon}{\ell}\left(1 - \left(\frac{q_{\mathrm{RO}}}{2^k} + \frac{q_{\mathrm{RO}}^2}{2^{2n}}\right)\right) \ge \frac{1}{\ell}\left(\epsilon - \frac{q_{\mathrm{RO}}}{2^k} - \frac{q_{\mathrm{RO}}^2}{2^{2n}}\right)\,.\end{aligned}$$

The running time of B is that of A plus at most 2ℓ evaluations of F. Equation (5) follows.

Possible tweaks. The scheme can be simplified not all seven properties need to be preserved. For example, if the key K is dropped from the input to RO_1, the $\mathcal{ROX}$ construction fails to preserve eSec and ePre, but still preserves all other notions. Dropping the message bits $\mathfrak{m}$ from the input of either RO_1 or RO_2 destroys the preservation of aSec and aPre, but leaves the preservation of other notions unharmed.

Acknowledgements

We would like to thank David Cash and the anonymous referees for their useful feedback. This work was supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT, and in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy). The first author is supported by a Ph.D. Fellowship and the second by a Postdoctoral Fellowship from the Flemish Research Foundation (FWO - Vlaanderen). The fourth author was supported by NSF CNS-0627752.

References

1. E. Andreeva, G. Neven, B. Preneel, and T. Shrimpton. Seven-property-preserving iterated hashing: ROX. Cryptology ePrint Archive, Report 2007/176, 2007.
2. E. Andreeva, G. Neven, B. Preneel, and T. Shrimpton. Three-property preserving iterations of keyless compression functions. ECRYPT Hash Workshop 2007.
3. M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *CRYPTO'96*, vol. 1109 of *LNCS*. Springer-Verlag, 1996.
4. M. Bellare and T. Ristenpart. Multi-property-preserving hash domain extension: The EMD transform. In *ASIACRYPT 2006*, vol. 4284 of *LNCS*, pages 299–314. Springer-Verlag, 2006.
5. M. Bellare and T. Ristenpart. Hash functions in the dedicated-key setting: Design choices and MPP transforms. In L. Arge, C. Cachin, and A. Tarlecki, editors, *34th International Colloquium on Automata, Languages and Programming – ICALP 2007*, vol. 4596 of *LNCS*. Springer-Verlag, 2007.
6. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93*, pages 62–73. ACM Press, 1993.
7. M. Bellare and P. Rogaway. Collision-resistant hashing: Towards making UOWHFs practical. In *CRYPTO'97*, vol. 1294 of *LNCS*. Springer-Verlag, 1997.
8. E. Biham and O. Dunkelman. A framework for iterative hash functions – HAIFA. Second NIST Cryptographic Hash Workshop, 2006.
9. J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In *CRYPTO 2005*, vol. 3621 of *LNCS*, pages 430–448. Springer-Verlag, 2005.
10. R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.
11. I. Damgård. A design principle for hash functions. In *CRYPTO'89*, vol. 435 of *LNCS*, pages 416–427. Springer-Verlag, 1990.

12. R. D. Dean. *Formal Aspects of Mobile Code Security.* PhD thesis, Princeton University, 1999.
13. S. Halevi and H. Krawczyk. Strengthening digital signatures via randomized hashing. In *CRYPTO 2006*, vol. 4117 of *LNCS*, pages 41–59. Springer-Verlag, 2006.
14. J. Kelsey and T. Kohno. Herding hash functions and the Nostradamus attack. In *EUROCRYPT 2006*, vol. 4004 of *LNCS*, pages 183–200. Springer-Verlag, 2006. Available from http://eprint.iacr.org/2005/281.
15. J. Kelsey and B. Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In *EUROCRYPT 2005*, vol. 3494 of *LNCS*, pages 474–490. Springer-Verlag, 2005.
16. X. Lai and J. L. Massey. Hash functions based on block ciphers. In *EUROCRYPT'92*, vol. 658 of *LNCS*, pages 55–70. Springer-Verlag, 1992.
17. W. Lee, D. Chang, S. Lee, S. H. Sung, and M. Nandi. New parallel domain extenders for UOWHF. In *ASIACRYPT 2003*, vol. 2894 of *LNCS*, pages 208–227. Springer-Verlag, 2003.
18. M. Luby and C. Rackoff. A study of password security. *Journal of Cryptology*, 1(3):151–158, 1989.
19. S. Lucks. A failure-friendly design principle for hash functions. In *ASIACRYPT 2005*, vol. 3788 of *LNCS*, pages 474–494. Springer-Verlag, 2005.
20. R. C. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Security and Privacy*, pages 122–134. IEEE Computer Society Press, 1980.
21. R. C. Merkle. A certified digital signature. In *CRYPTO'89*, vol. 435 of *LNCS*, pages 218–238. Springer-Verlag, 1990.
22. R. C. Merkle. One way hash functions and DES. In *CRYPTO'89*, vol. 435 of *LNCS*, pages 428–446. Springer-Verlag, 1990.
23. I. Mironov. Hash functions: From Merkle-Damgård to Shoup. In *EUROCRYPT 2001*, vol. 2045 of *LNCS*, pages 166–181. Springer-Verlag, 2001.
24. I. Mironov. Collision-resistant no more: Hash-and-sign paradigm revisited. In *PKC 2006*, LNCS, pages 140–156. Springer-Verlag, 2006.
25. M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *21st ACM STOC*, pages 33–43. ACM Press, 1989.
26. J. B. Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *CRYPTO 2002*, vol. 2442 of *LNCS*, pages 111–126. Springer-Verlag, 2002.
27. P. Rogaway. Formalizing human ignorance: Collision-resistant hashing without the keys. In *Vietcrypt 2006*, vol. 4341 of *LNCS*. Springer-Verlag, 2006.
28. P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *FSE 2004*, vol. 3017 of *LNCS*, pages 371–388. Springer-Verlag, 2004.
29. P. Sarkar. Masking-based domain extenders for UOWHFs: bounds and constructions. *IEEE Transactions on Information Theory*, 51(12):4299–4311, 2005.
30. V. Shoup. A composition theorem for universal one-way hash functions. In *EUROCRYPT 2000*, vol. 1807 of *LNCS*, pages 445–452. Springer-Verlag, 2000.
31. D. Wagner and I. Goldberg. Proofs of security for the Unix password hashing algorithm. In *ASIACRYPT 2000*, vol. 1976 of *LNCS*. Springer-Verlag, 2000.
32. X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In *CRYPTO 2005*, vol. 3621 of *LNCS*, pages 17–36. Springer-Verlag, 2005.
33. X. Wang and H. Yu. How to break MD5 and other hash functions. In *EUROCRYPT 2005*, vol. 3494 of *LNCS*, pages 19–35. Springer-Verlag, 2005.