Three-Property Preserving Iterations of Keyless Compression Functions

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

[1] SCD-COSIC, Dept. of Electrical Engineering, Katholieke Universiteit Leuven
Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium
{Elena.Andreeva,Gregory.Neven,Bart.Preneel}@esat.kuleuven.be
[2] Dept. of Computer Science, Portland State University
1900 SW 4th Avenue, Portland, OR 97201, USA
teshrim@cs.pdx.edu

Abstract

Almost all hash functions are based on the Merkle-Damgård iteration of a finite-domain compression function. It has been shown that this iteration preserves collision resistance, but it does not preserve other properties such as preimage or second preimage resistance. The recently proposed ROX construction provably preserves all seven security notions put forward by Rogaway and Shrimpton at FSE 2004, but it does so for families of hash functions, that is, the compression function is indexed by a public parameter known as a key. Practical hash functions however do not have such a parameter, so it is not entirely clear how to instantiate these schemes. We use Rogaway's human-ignorance approach to resolve this situation, and present four different iterations (two of them chaining-based, two of them tree-based) that provably preserve all three notions of collision, preimage and second preimage resistance.

Keywords: Cryptographic hash functions, Merkle-Damgård, ROX, collision resistance, preimage resistance, second preimage resistance, provable security.

1 Introduction

A cryptographic hash function maps messages of arbitrary length to a fixed-length hash value that acts as a "fingerprint" for the message. Their main security requirement is collision resistance, meaning that it should be hard to find two different messages with the same hash value. Practical hash functions today are almost always constructed by iterating a finite-domain compression function, typically using the Merkle-Damgård chaining method [Mer90, Dam90]. The main feature of the construction is that it *preserves* collision resistance, meaning that if the compression function is collision resistant, then the iterated hash function is collision resistant as well.

But collision resistance may not be the only property that we are interested in. Second-preimage resistance for example requires that it is hard to, given a random message, find a second message that has the same hash value. This is the type of security required when using hash values to guard integrity of data, for example of files stored on a hard disk. Also, a variant of second preimage resistant hash functions can be used to build secure hash-and-sign signature schemes [BR97]. A third property one may require is preimage resistance, meaning that given a random hash value, it should be hard to come up with a message that hashes to this value. Preimage resistance is the type of property needed for the protection of stored passphrases.

Rogaway and Shrimpton [RS04] studied seven different security notions for hash functions and the relations among them. Their notions include the three main ones of collision (Coll), second preimage

(Sec) and preimage (Pre) resistance, and *everywhere* and *always* variants of the latter two (eSec, aSec, ePre, aPre). They show that Coll directly implies Sec, and that Sec implies Pre if the hash function is sufficiently compressing (which is the case for all practical purposes). When iterating a collision resistant compression function using the Merkle-Damgård construction, these implications show that the resulting hash function is also second preimage and preimage resistant.

As recently argued by [ANPS07] however, this may not be satisfactory. In the light of recent collision attacks on hash functions [WY05, WYY05], designing collision resistant hash functions seems to be a highly non-trivial task. When the compression function is not collision resistant, then all guarantees for the iterated function are off, as was illustrated by the second preimage attacks of Kelsey and Schneier [KS05]. From a quantitative point of view, for a hash function with n-bit output one can only expect to have collision resistance up to $2^{n/2}$ time steps due to birthday attacks, but one would hope to have (second) preimage resistance up to 2^n time steps. Since Merkle-Damgård inherits its (second) preimage resistance from collision resistance, it can only guarantee security up to $2^{n/2}$ time steps.

As a solution to this problem, the recently proposed ROX iteration [ANPS07] uses a small-input random oracle to natively preserve all seven security notions of [RS04]. (The compression function itself is still modeled as a real function, not as a random oracle.) An important discrepancy between theory and practice of hash functions however is that to properly define collision resistance, one has to look at hash functions as occurring in families, of which members are indexed by a publicly known key. In practice, it is hard to see which part of the description of SHA-256 can be used as a key, or even to describe the family it belongs to. This discrepancy was recently solved by Rogaway [Rog06], who suggests to formulate theorems so that they prescribe an explicit reduction, rather than the non-existence of an efficient algorithm.

Our contributions. The goal of this paper is to find and prove secure iterations of *keyless* compression functions that natively preserve the three main notions of collision and (second) preimage resistance. The iterations themselves can be keyed, as long as it is clear how to sample random keys. This accommodates applications that need to sample a random member of a family of hash functions, such as hash-and-sign signatures [BR97]. For applications that only need a single hash function, a random key can be fixed in a standard.

First, it is worth noting that, as is the case for keyed compression functions [ANPS07], there exist counterexamples of Sec and Pre secure keyless compression functions that, when iterated using Merkle-Damgård, are completely insecure. We revisit an iteration of keyless compression functions due to Shoup [Sho00], for which he proved that if the compression function is Sec secure, then the iterated hash function is eSec secure. The notion of eSec, sometimes also referred to as universal one-way (UOWHF) or target collision resistant (TCR) hash functions, directly implies that of Sec, and is of practical importance for use in signatures. Note that the iteration thereby achieves an even stronger property than mere preservation, namely "promotion" of Sec security to the stronger eSec notion. We prove that the $\mathcal{SH}$ iteration, as we call it here, also preserves collision resistance (in the sense of [Rog06]) and Pre security.

Next, we propose the *modified XOR-Tree hash* ($\mathcal{MXT}$), which is an adaptation of the XOR-Tree hash of [BR97], and prove that it also preserves all three notions of Coll, Sec, and Pre. As was the case for the $\mathcal{SH}$ iteration, we even prove a stronger result that Sec security of the compression function yields eSec security for the hash function. While often considered less desirable than chaining constructions because of their higher memory usage, tree constructions have the advantage of being naturally parallelizable, making them perfectly suited for hashing large amounts of static data.

Both the $\mathcal{SH}$ and the $\mathcal{MXT}$ hashes have the disadvantage of having a key length that is logarithmic in the message length. Especially for the application to signatures this is problematic, as these require to sign the key along with the message, and the underlying signature scheme may not be able to

accommodate for this. We therefore apply a technique from [ANPS07] to achieve constant (say, 128-bit) key length by generating the other keys through a random oracle. This random oracle has a small input size (around 134 bits) and is applied only a logarithmic number of times (in the message length), and therefore could be instantiated using a primitive like AES that is considered too expensive to apply a linear number of times.

What about other properties? The security notions formalized by [RS04] are certainly not the only ones of interest. Bellare and Ristenpart [BR06], following previous work by Coron et al. [CDMP05] and Bellare et al. [BCK96], 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. Kelsey and Kohno [KK06] suggest chosen-target forced-prefix security as the right goal to stop Nostradamus attacks. We leave the study of the preservation of these properties by the investigated in this paper constructions to future work.

2 Security Definitions

Notation. In this section, we explain the security notions for families of hash functions of [RS04]. 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. The empty string is denoted ε. If x is a string and $i \in \mathbb{N}$, then $x^{(i)}$ is the i-th bit 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 of compression functions. In this work we consider fixed-input-size compression functions to be unkeyed, modeling most closely real-world compression functions such as SHA-256, and we assume arbitrary-input-size hash functions are families of functions indexed by keys. We first give advantage definitions for collision (Coll), second preimage (Sec), and preimage resistance (Pre) for the former. A compression function is a function $\mathrm{F} : \mathcal{M} \to \mathcal{Y}$ where $\mathcal{M}$ and $\mathcal{Y}$ are finite sets of bit strings. For a fixed adversary A and $\lambda \in \mathbb{N}$, we define the following advantage functions:

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{Coll}}_{\mathrm{F}}(\mathsf{A}) &= \Pr\left[\; M', M \xleftarrow{\$} \mathsf{A}(\varepsilon) \;:\; M \neq M' \text{ and } \mathrm{F}(M) = \mathrm{F}(M') \;\right] \\
\mathbf{Adv}^{\mathrm{Sec}[\lambda]}_{\mathrm{F}}(\mathsf{A}) &= \Pr\left[\; M \xleftarrow{\$} \{0,1\}^\lambda \,;\, M' \xleftarrow{\$} \mathsf{A}(M) \;:\; M \neq M' \text{ and } \mathrm{F}(M) = \mathrm{F}(M') \;\right] \\
\mathbf{Adv}^{\mathrm{Pre}[\lambda]}_{\mathrm{F}}(\mathsf{A}) &= \Pr\left[\; M \xleftarrow{\$} \{0,1\}^\lambda \,;\, Y \leftarrow \mathrm{F}(M) \,;\, M' \xleftarrow{\$} \mathsf{A}(Y) \;:\; \mathrm{F}(M') = Y \;\right]
\end{aligned}$$

The compression function F is said to be (t, ϵ) Sec or Pre secure if $\mathbf{Adv}^{\mathrm{Sec}[\lambda]}_{\mathrm{F}}(\mathsf{A}) < \epsilon$ or $\mathbf{Adv}^{\mathrm{Pre}[\lambda]}_{\mathrm{F}}(\mathsf{A}) < \epsilon$ for all adversaries A running in time at most t and for all $\lambda \in \mathbb{N}$ such that $\{0,1\}^\lambda \subseteq \mathcal{M}$. Note that it is impossible to define security for the case of Coll in an analogous way. Indeed, if collisions on F exist, then an adversary A that simply prints out a collision that is hardcoded into it always has advantage 1. Rather than defining Coll security through the non-existence of an algorithm A, we follow Rogaway's human-ignorance approach [Rog06] and use the above advantage function as a metric to

relate the advantage of an adversary A against the hash function to that of an adversary B against the compression function.

When giving results in the random oracle model, we also account for the total number of queries q_{RO} that the adversary makes to its random oracles. In this case, we will write $(t, q_{\mathrm{RO}}, \epsilon)$-secure with the obvious meaning. If the hash function uses multiple random oracles, we define q_{RO} to be the sum of the number of queries that the adversary can make to each separately.

Security of hash functions. 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 $\{0,1\}^\lambda \subseteq \mathcal{M}$. Following [RS04], we use the following advantage measures:

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{Coll}}_{\mathrm{H}}(\mathsf{A}) &= \Pr\left[K \xleftarrow{\$} \mathcal{K}\,;\, (M, M') \xleftarrow{\$} \mathsf{A}(K) \,:\, \begin{array}{c} M \neq M' \text{ and} \\ \mathrm{H}(K, M) = \mathrm{H}(K, M') \end{array} \right] \\
\mathbf{Adv}^{\mathrm{Sec}[\lambda]}_{\mathrm{H}}(\mathsf{A}) &= \Pr\left[\begin{array}{c} K \xleftarrow{\$} \mathcal{K}\,;\, M \xleftarrow{\$} \{0,1\}^\lambda \\ M' \xleftarrow{\$} \mathsf{A}(K, M) \end{array} \,:\, \begin{array}{c} M \neq M' \text{ and} \\ \mathrm{H}(K, M) = \mathrm{H}(K, M') \end{array} \right] \\
\mathbf{Adv}^{\mathrm{eSec}}_{\mathrm{H}}(\mathsf{A}) &= \Pr\left[\begin{array}{c} (M, St) \xleftarrow{\$} \mathsf{A}\,;\, K \xleftarrow{\$} \mathcal{K} \\ M' \xleftarrow{\$} \mathsf{A}(K, St) \end{array} \,:\, \begin{array}{c} M \neq M' \text{ and} \\ \mathrm{H}(K, M) = \mathrm{H}(K, M') \end{array} \right] \\
\mathbf{Adv}^{\mathrm{Pre}[\lambda]}_{\mathrm{H}}(\mathsf{A}) &= \Pr\left[\begin{array}{c} K \xleftarrow{\$} \mathcal{K}\,;\, M \xleftarrow{\$} \{0,1\}^\lambda \\ Y \leftarrow \mathrm{H}(K, M)\,;\, M' \xleftarrow{\$} \mathsf{A}(K, Y) \end{array} \,:\, \mathrm{H}(K, M') = Y \right]
\end{aligned}$$

These are the standard three notions of *collision-resistance* (Coll), *preimage resistance* (Pre) and *second-preimage resistance* (Sec), and the stronger variant of *everywhere second-preimage resistance* (eSec). For $\mathrm{atk} \in \{\mathrm{Coll}, \mathrm{eSec}\}$, we say that H is (t, ϵ) atk secure if $\mathbf{Adv}^{\mathrm{atk}}_{\mathrm{H}}(\mathsf{A}) < \epsilon$ for all adversaries A running in time at most t. For $\mathrm{atk} \in \{\mathrm{Sec}, \mathrm{Pre}\}$, we say that H is (t, ϵ) atk secure if $\mathbf{Adv}^{\mathrm{atk}[\lambda]}_{\mathrm{H}}(\mathsf{A}) < \epsilon$ for all adversaries A running in time at most t and for all $\lambda \in \mathbb{N}$ such that $\{0,1\}^\lambda \subseteq \mathcal{M}$.

Security preservation. Our goal is to build an infinite-domain hash function family H out of a limited-domain compression function F so that the hash function "inherits" its Coll, Sec, and Pre security from the natural analogues of these properties for F. For $\mathrm{atk} \in \{\mathrm{Sec}, \mathrm{Pre}\}$, we say that H *preserves* atk security if H is (t, ϵ) atk secure whenever F is (t', ϵ') atk secure, for some well-specified relation between $t, t', \epsilon, \epsilon'$. For the case of Coll, we have to be more careful because, as pointed out before, (t, ϵ)-Coll security cannot be defined for the keyless compression function F. Rather, we follow Rogaway [Rog06] by saying that collision resistance is preserved if, for an explicitly given Coll adversary A against H, there exists a corresponding, explicitly specified Coll adversary B, as efficient as A, that finds collisions for H.

3 A Chaining Construction

Rogaway [Rog06] showed that the Merkle-Damgård iteration preserves collision resistance for keyless compression functions. However, one can come up with counterexamples (see [ANPS07] and Appendix B) to demonstrate that it does not preserve second-preimage or preimage resistance.

Shoup [Sho00] presented a keyed iteration of keyed compression functions that preserves eSec security, and at the end of the paper also showed how a keyed eSec secure compression function can be constructed from an unkeyed Sec secure one. Since eSec security implies Sec security [RS04], this instantiation of Shoup's iteration yields the first, and to the best of our knowledge only known iteration

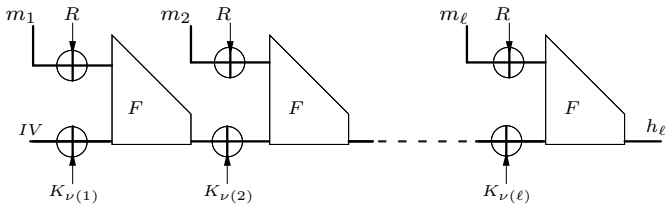

Figure 1: **The Shoup's Hash with Keyless Compression Function.**

of an unkeyed compression function that preserves Sec security. In fact, it achieves an even stronger property because it "promotes" Sec security of the compression function to eSec security for the hash function. This is important for hash-and-sign signatures [BR97].

We revisit Shoup's iteration here instantiated with a keyless compression function F as prescribed in [Sho00]. For a compression function $\mathrm{F} : \{0,1\}^{b+n} \to \{0,1\}^n$, a fixed initialization vector $IV \in \{0,1\}^n$, b-bit key R and n-bit keys $K_0 \| \ldots \| K_{l_{\max}}$, it is given by

Algorithm $\mathcal{SH}_{\mathrm{F}}(R \| K_0 \| \ldots \| K_{l_{\max}}, M)$:
$m_1 \| \ldots \| m_\ell \leftarrow$ ls-pad(M) ; $h_0 \leftarrow IV$
For $i = 1, \ldots, \ell$ do $h_i \leftarrow F(m_i \oplus R \| h_{i-1} \oplus K_{\nu(i)})$
Return h_ℓ .

A graphical representation of Shoup's hash is given in Figure 1. Here, $l_{\max} = \lceil \log_2(\Lambda/b) \rceil$, where Λ is the maximum message length; ls-pad(M) splits M up in ℓ blocks of size b by padding the message M with bits $10 \ldots 0$ up to the next multiple of b and adding a length strengthening block $\langle M \rangle_b$; and $\nu(i)$ is the largest integer j such that $2^j \mid i$.
In the following, we show that apart from second-preimage resistance, the $\mathcal{SH}_{\mathrm{F}}$ iteration also preserves collision and preimage resistance.

Theorem 3.1 If there exists an explicitly given adversary A that (t, ϵ)-breaks the Coll security of $\mathcal{SH}_{\mathrm{F}}$, then there exists an explicitly given adversary B that (t', ϵ')-breaks the Coll security of F for $\epsilon' \geq \epsilon$ and $t' \leq t + 2\ell \cdot \tau_{\mathrm{F}}$, where τ_{F} is the time needed for one evaluation of F, $\ell = \lceil \lambda/b \rceil + 1$, and λ is the maximum length of the two messages output by A.

Proof: Given a Coll-adversary A against $\mathcal{SH}_{\mathrm{F}}$, we construct the Coll adversary B against F. First B chooses random key strings $R \xleftarrow{\$} \{0,1\}^b$ and $K_0, \ldots, K_{l_{\max}} \xleftarrow{\$} \{0,1\}^n$. B runs A on input $K = R \| K_0 \| \ldots \| K_{l_{\max}}$ to obtain a pair of colliding messages M and M'. Let $m_1 \| \ldots \| m_\ell \leftarrow$ ls-pad(M), $m'_1 \| \ldots \| m'_{\ell'} \leftarrow$ ls-pad(M'), and let $h_0, \ldots, h_\ell$ and $h'_0, \ldots, h'_{\ell'}$ be the intermediate hash values obtained in the $\mathcal{SH}$ computation of M and M', respectively.

We know that $h_\ell = h'_{\ell'}$. If $|M| \neq |M'|$, then $m_\ell \neq m'_{\ell'}$ and B outputs $(m_\ell \oplus R \| h_{\ell-1} \oplus K_{\nu(\ell)})$ and $(m'_{\ell'} \oplus R \| h'_{\ell'-1} \oplus K_{\nu(\ell')})$ as its colliding pair. If $h_{\ell-1} \oplus K_{\nu(\ell)} \neq h'_{\ell'-1} \oplus K_{\nu(\ell')}$ then B outputs the same pair as its collision.

Otherwise, we have that $|M| = |M'|$ and therefore that $\ell = \ell'$. If $m_{\ell-1} \neq m'_{\ell-1}$ or $h_{\ell-2} \oplus K_{\nu(\ell-1)} \neq h'_{\ell-2} \oplus K_{\nu(\ell-1)}$, then B outputs $(m_{\ell-1} \oplus R) \| (h_{\ell-2} \oplus K_{\nu(\ell-1)})$ and $(m'_{\ell-1} \oplus R) \| (h'_{\ell-2} \oplus K_{\nu(\ell-1)})$ as its colliding messages. Otherwise, we have that $h_{\ell-2} = h'_{\ell-2}$, so we can repeat the same reasoning for the case that $m_{\ell-2} \neq m'_{\ell-2}$ or $h_{\ell-3} \oplus K_{\nu(\ell-2)} \neq h'_{\ell-3} \oplus K_{\nu(\ell-2)}$. Proceeding this way from right to left throughout the chain, one can see that at some point B will find a collision on F, unless $M = M'$. ∎

Theorem 3.2 If F is (t', ϵ') Pre-secure, then $\mathcal{SH}_{\mathrm{F}}$ is (t, ϵ) Pre-secure for $\epsilon \geq \epsilon'$ and $t \leq t' - \ell \cdot \tau_{\mathrm{F}}$, where τ_{F} is the time required for an evaluation of F, $\ell = \lceil \lambda/b \rceil$, and λ is the maximum message length

Proof: Given a Pre[λ]-adversary A against $\mathcal{SH}_{\mathrm{F}}$, consider the following Pre-adversary B against F. B gets as input a random target value Y. It chooses random keys $R \xleftarrow{\$} \{0,1\}^b$ and $K_1, \ldots, K_{l_{\max}} \xleftarrow{\$} \{0,1\}^n$, and runs A on input $K = R\|K_0\| \ldots \|K_{l_{\max}}$ and Y until it outputs a preimage M'. Let $m'_1\| \ldots \|m'_{\ell'} \leftarrow$ ls-pad(M') and let $h'_0, \ldots, h'_{\ell'}$ be the intermediate hash values obtained when computing $\mathcal{SH}_{\mathrm{F}}(K, M')$ as described above. Algorithm B outputs $(m'_{\ell'} \oplus R)\|(h'_{\ell'-1} \oplus K_{\nu(\ell')})$ as its own preimage.

It is important to realize here that, due to the random choice of the keys, the target point Y given as input to B is also correctly distributed as an input for A. Namely, Y follows the distribution induced by applying F to a random input from $\{0,1\}^{b+n}$. Algorithm A expects a target point Y that is the output of $\mathcal{SH}_{\mathrm{F}}(K, M)$ for a random key K and a random message M. The final message block m_ℓ and intermediate hash $h_{\ell-1}$ are not necessarily random (in fact, $m_\ell = |M|$ is non-random), but due to the XOR with fresh keys R and $K_{\nu(\ell)}$ the input to the last compression function are indeed random. ∎

4 A Tree Construction

A major disadvantage of chaining constructions is that the computations cannot be parallelized. Indeed, the next compression function cannot be evaluated until the output of the previous one is known. For applications where large amounts of data have to be hashed using parallel processors, tree constructions can be 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 and later combined to compute the final hash value.

The simplest tree iteration is the (strengthened) Merkle tree [Mer80]. It can be shown to preserve collision resistance of the underlying compression function, but counterexamples similar to those of [ANPS07] can be used to show that second-preimage and preimage resistance are not preserved. A second candidate is the XOR-Tree construction of [BR97]. We show in Appendix C that this iteration preserves preimage resistance, but were unable to show or contradict the preservation of Coll and Sec. We managed to prove that the XOR-Tree hash is collision resistant if the compression function satisfies the stronger property of δ-collision resistance, but this is not the "pur-sang" preservation of Coll that we are looking for. (See Appendix C for details.)

We therefore describe a slightly modified construction called the *modified XOR-Tree* (mXT) hash here, and prove that it preserves all three notions of Coll, Sec, and Pre. First we introduce some additional notation for tree hashes. With a we denote the arity of tree hash structures, that is the number of inputs to a node, equivalent here to number of equal length inputs to the compression function. The depth of the tree is denoted by d and the enumeration of tree levels starts from index one, the widest part of the tree (level with maximal number of nodes), to reach the level d at the root of the tree. For a compression function $\mathrm{F} : \{0,1\}^{an} \to \{0,1\}^n$, the mXT scheme proceeds as follows:

Algorithm $\mathit{mXT}_{\mathrm{F}}(K_1\| \ldots \|K_{d_{\max}}\|K^*\ ,\ M)$:
  $m_1\| \ldots \|m_{a^d} \leftarrow$ tpad(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
    $h_{i,j} \leftarrow \mathrm{F}\left((h_{i-1,(j-1)a+1}\| \ldots \|h_{i-1,ja}) \oplus K_i\right)$
  $h_{d+1,1} \leftarrow \mathrm{F}\left((h_{d,1}\|\langle|M|\rangle_{n(a-1)}) \oplus K^*\right)$
  Return $h_{d+1,1}$

We provide the graphical description of the modified XOR-Tree in Figure 2. Here, $d_{\max} = \lceil \log_a \Lambda \rceil$ where Λ is the maximum message length, and tpad is a padding function that appends bits $10 \ldots 0$ to

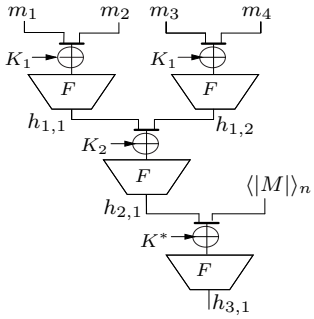

Figure 2: **The Modified XOR-Tree Hash with Keyless Compression Function.** We illustrate a tree of depth $d = 2$ and arity $a = 2$.

the end of M until it has length a^d for some integer $d \in \mathbb{N}$. The only difference with the original XOR-Tree scheme is that here, the key K^* used before the final application of the compression function is always the same. That means K^* is independent of the tree depth, while in the original scheme it would simply be the next key in the sequence K_{d+1}.

The following theorems show that the modified XOR-Tree iteration preserves all three of Coll, Sec, and Pre. In fact, Theorem 4.2 proves an even stronger result, analogously to the case of $\mathcal{SH}^{\mathrm{F}}$. Namely, it shows that if F is Sec secure, then the iterated hash $m\mathcal{XT}^{\mathrm{F}}$ achieves the stronger notion of eSec security. Similarly to the $\mathcal{SH}^{\mathrm{F}}$, the Sec preservation of the $m\mathcal{XT}^{\mathrm{F}}$ is implied by the eSec result.

Theorem 4.1 If there exists an explicitly given adversary A that (t, ϵ)-breaks the Coll security of $m\mathcal{XT}^{\mathrm{F}}$, then there exists an explicitly given adversary B that (t', ϵ')-breaks the Coll security of F for $\epsilon' \geq \epsilon$ and $t' \leq t + 2(\frac{a^d-1}{a-1} + 1) \cdot \tau_{\mathrm{F}}$, where τ_{F} is the time needed for one evaluation of F, where d is the smallest integer such that $a^d \geq \lambda$, and where λ is the maximum message length.

Proof: Given a Coll adversary A against $m\mathcal{XT}^{\mathrm{F}}$, we build the following Coll adversary B against F. B generates random keys $K_1, \ldots, K_{d_{\max}}, K^* \xleftarrow{\$} \{0,1\}^{an}$ and runs A on input $K = K_1 \| \ldots \| K_{d_{\max}} \| K^*$ until it outputs a pair of colliding messages M and M'. Let $m_1 \| \ldots \| m_\ell \leftarrow \mathrm{tpad}(M)$, $m'_1 \| \ldots \| m'_{\ell'} \leftarrow \mathrm{tpad}(M')$, and $h_{d,1}$ and $h'_{d',1}$ be the one-but-last output hash values computed in the execution of $m\mathcal{XT}^{\mathrm{F}}(M)$ and $m\mathcal{XT}^{\mathrm{F}}(M')$ respectively.

If $|M| \neq |M'|$, then $\big(h_{d,1} \| \langle |M| \rangle_{(a-1)n}\big) \oplus K^*$ and $\big(h'_{d',1} \| \langle |M'| \rangle_{(a-1)n}\big) \oplus K^*$ are clearly different, so this pair of inputs to F is a valid colliding pair for B. Also, if $h_d \neq h'_{d'}$, then this pair of messages forms a valid colliding pair for B.

Otherwise, we know that $|M| = |M'|$ and therefore that $d' = d$. If $h_{d-1,j} \neq h'_{d-1,j}$ for some $1 \leq j \leq a$, then the inputs $\big(h_{d-1,1} \| \ldots \| h_{d-1,a}\big) \oplus K_d$ and $\big(h'_{d-1,1} \| \ldots \| h'_{d-1,a}\big) \oplus K_d$ form a valid colliding pair for F. Continuing this argument, B can proceed bottom-up throughout the tree until a colliding pair of inputs is found, unless $M = M'$. The running time of B equals that of A plus up to $2(\frac{a^d-1}{a-1} + 1)$ evaluations of F. ▌

Theorem 4.2 If F is (t', ϵ') Sec secure, then $m\mathcal{XT}^{\mathrm{F}}$ is (t, ϵ) eSec secure for $\epsilon \geq (\frac{a^d-1}{a-1} + 1)\epsilon'$ and $t \leq t' - 2(\frac{a^d-1}{a-1} + 1) \cdot \tau_{\mathrm{F}}$, where τ_{F} is the time needed for an evaluation of F, where λ is the maximum message length, and where d is the smallest integer such that $a^d \geq \lambda$.

Proof: Given a eSec-adversary A against $m\mathcal{XT}^{\mathrm{F}}$, consider the following Sec-adversary B against F. B is given as input a random message $x = x_1 \| \ldots \| x_a \in \{0,1\}^{an}$ where $|x_i| = n$ for $i = 1 \ldots a$. B

runs A to obtain A's target message M. Let $\lambda = |M|$ and $m_1 \| \ldots \| m_\ell \leftarrow \text{tpad}(M)$. The goal of B is to correctly embed its challenge message x in the $m\mathcal{XT}$ iteration and B achieves this in the choice of key sequence for A. Hence B selects a random index $i \xleftarrow{\$} \{1, \ldots, (\frac{a^d-1}{a-1}+1)\}$. If $i \in \{1, \ldots, \ell\}$, then B sets $K_1 = x \oplus (m_{(i-1)a+1} \| \ldots \| m_{ai})$ and chooses the rest of the keys $K_2, \ldots, K_{d_{\max}}, K^*$ at random. If $i \in \{\ell, \ldots, \frac{a^d-1}{a-1}\}$, then B chooses at random keys $K_1, \ldots, K_{j-1}$ and $K_{j+1}, \ldots, K_{d_{\max}}, K^*$ and sets $K_j = x \oplus (h_{j,(k-1)a+1} \| \ldots \| h_{j,ka})$. We compute j as the smallest integer, such that the inequality $\sum_{y=1}^{j} a^{d-1-y} \geq i$ is satisfied, or $j = \lceil (d - \log_a (i(1-a)+\ell)) \rceil$, and $k = i - \sum_{y=d-j+1}^{d-1} a^{d-1-y}$. Finally, if $i = \frac{a^d-1}{a-1} + 1$, then B chooses at random keys $K_1, \ldots, K_{d_{\max}}$ and sets the value of $K^* = x \oplus (h_{d,1} \| \langle |M| \rangle_{(a-1)n})$.

Next, A obtains the generated keys $K_1 \| \ldots \| K_{d_{\max}} \| K^*$ from B and outputs its colliding second preimage message M'. Identically to the proof in Theorem 4.1, B searches for colliding inputs to the compression function F. With probability $\frac{a-1}{a^d-a-2}$ B finds the colliding pair at the correct position i (at which its challenge message x was embedded) and outputs the colliding F input as its valid second preimage. ∎

Theorem 4.3 If F is (t', ϵ') Pre-secure, then $m\mathcal{XT}_{\text{F}}$ is (t, ϵ) Pre-secure for $\epsilon \geq \epsilon'$ and $t \leq t' - (\frac{a^d-1}{a-1} + 1) \cdot \tau_{\text{F}}$, where τ_{F} is the time needed for an evaluation of F, where λ is the length of the message output by A, and where d is the smallest integer such that $a^d \geq \lambda$.

Proof: Given a Pre[λ]-adversary A against $\mathcal{XT}_{\text{F}}$, consider the following Pre-adversary B against F. B obtains a target value Y (computed for a random message input $x = x_1 \| \ldots \| x_a$ of an-bits) and generates at random $an(d+1)$-bit sequence of key strings $K_1 \| \ldots \| K_{d_{\max}} \| K^*$. B runs A on the same target value Y and the generated key sequence. A returns to B its preimage message M'. Let $m'_1 \| \ldots \| m'_\ell \leftarrow \text{tpad}(M')$ and let $h'_{d,1}$ be the one-but-last output hash value computed in an execution of $m\mathcal{XT}_{\text{F}}(K_1 \| \ldots \| K_d \| K^*, M')$. Algorithm B outputs $(h'_{d,1} \| \langle |M| \rangle_{(a-1)n} \oplus K^*)$ as its own preimage. Notice that the output distributions induced by the random choice of the input x to F and M' to $m\mathcal{XT}_{\text{F}}$ do not differ in a similar way to the Pre-security (Theorem 3.2) of $\mathcal{SH}$. ∎

5 Shorter Keys using Random Oracles

A major disadvantage of Shoup's and the XOR-Tree hashes is that they require keys that are logarithmic in the message length. While this is of course much better than key length linear in the message length, it can still be problematic for certain applications. For example, when using the construction of hash-and-sign signatures from eSec secure hash functions of [BR97] in combination with DSA, the hash value and the keys together have to be encoded as an integer modulo the group order. For a security level of 128 bits, in principle one needs a group order of size 256 bits for the discrete logarithm problem to be hard. However, if we want to sign messages up to 2^{20} blocks, then the hash value and the keys together take $128 + 20 \cdot 128 = 2688$ bits. (Note that to get 128 bits of eSec security, a hash function with 128-bit output can suffice.) Since exponentiation is a cubic operation, increasing the group order to this size makes exponentiations $(2688/256)^3 \approx 1158$ times more expensive!

In this section, we use a technique from [ANPS07] and show how the key length can be further reduced to 128 bits by having the keys generated through a random oracle. In the application to hash-and-sign signature sketched above, this means that the hash and key fit nicely within the space of a 256-bit group order.

While it may be controversial to use a random oracle in a hash iteration, we stress that it is *not* the compression function itself that is modeled as a random oracle. Moreover, the random oracle used in the iteration has a very limited input size, typically around 134 bits to achieve a 128-bit security level.

5.1 Shorter Keys for Shoup's Hash

Let $\mathrm{F} : \{0,1\}^{b+n} \to \{0,1\}^n$ be a compression function, let $\mathrm{RO}_1 : \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^n$ and $\mathrm{RO}_2 : \{0,1\}^k \to \{0,1\}^b$ be random oracles where $l = \lceil \log_2 \log_2 \Lambda/b \rceil$, and let the functions $\nu(\cdot)$ and ls-pad$(\cdot)$ are defined as for the $\mathcal{SH}_{\mathrm{F}}$ construction. The *random-oracle Shoup* iteration $r\mathcal{SH}_{\mathrm{F}}$ is given by

Algorithm $r\mathcal{SH}_{\mathrm{F}}(K, M)$:
$m_1 \| \ldots \| m_\ell \leftarrow$ ls-pad(M); $h_0 \leftarrow IV$
For $i = 0, \ldots, \lceil \log_2 \ell \rceil$ do $K_i \leftarrow \mathrm{RO}_1(K, \langle i \rangle_l)$
$R \leftarrow \mathrm{RO}_2(K)$
For $i = 1, \ldots, \ell$ do $h_i \leftarrow \mathrm{F}(m_i \oplus R \| h_{i-1} \oplus K_{\nu(i)})$
Return h_ℓ .

The theorems below state that the random-oracle Shoup iteration preserves collision, second-preimage, and preimage resistance in the random oracle model. The proofs are quite similar to those of the standard Shoup iteration. Rather than reproving the full scheme here, we sketch how the proofs differ from their analogues for $\mathcal{SH}_{\mathrm{F}}$.

Theorem 5.1 If there exists an explicitly given adversary A that $(t, q_{\mathrm{RO}}, \epsilon)$-breaks the Coll security of $r\mathcal{SH}_{\mathrm{F}}$ when $\mathrm{RO}_1, \mathrm{RO}_2$ are modeled as random oracles, then there exists an explicitly given adversary B that (t', ϵ')-breaks the Coll security of F for $\epsilon' \geq \epsilon$ and $t' \leq t + q_{\mathrm{RO}} + 2\ell \cdot \tau_{\mathrm{F}}$, where τ_{F} is the time needed for one evaluation of F, $\ell = \lceil \lambda/b \rceil + 1$, and λ is the maximum length of the two messages output by A.

Proof: (Sketch) The only difference with the proof of Theorem 3.1 is in the way B generates the keys. Instead of selecting random strings $R, K_0, \ldots, K_{l_{\max}}$, now B simply generates a random string $K \xleftarrow{\$} \{0,1\}^k$. It then runs A on input K, responding to its random oracle queries as usual, i.e. by returning random values for new queries and being consistent for previously asked queries. In estimate for the running time t, we assume that responding a random oracle query this way takes unit time. ∎

Theorem 5.2 If F is a (t', ϵ') Sec secure compression function, then $r\mathcal{SH}_{\mathrm{F}}$ is a $(t, q_{\mathrm{RO}}, \epsilon)$ eSec secure hash function in the random oracle model for $\epsilon \geq \ell\epsilon' + q_{\mathrm{RO}}/2^k$ and $t \leq t' - q_{\mathrm{RO}} - 2\ell \cdot \tau_{\mathrm{F}}$. Here, RO_1 and RO_2 are modeled as random oracles, τ_{F} is the time required for an evaluation of F, $\ell = \lceil \lambda/b \rceil$ and λ is the maximum message length.

Proof: Given a eSec-adversary A against $r\mathcal{SH}_{\mathrm{F}}$, consider the following Sec-adversary B against F. B gets as input a random $b+n$-bits target message $x = x_1 \| x_2$ ($|x_1| = b$ and $|x_2| = n$). Then B runs A to obtain its target message M. Let $m_1 \| \ldots \| m_\ell \leftarrow$ ls-pad(M). Then B generates a k-bit key K at random. B simulates the answers to A's random oracle queries RO_1 and RO_2 by running algorithm $\mathsf{RO\text{-}Sim}_1$ and $\mathsf{RO\text{-}Sim}_2$ as described in Appendix A. It also maintains associative arrays $T_1[\cdot]$ and $T_2[\cdot]$.

B then chooses at random an index $i \xleftarrow{\$} \{1 \ldots \ell\}$ and embeds its x_1 and x_2 target values in the $r\mathcal{SH}$ iteration of M by preprogramming the values of R and $K_{\nu(i)}$ in T_2 and T_1 respectively. To enforce that $R = m_i \oplus x_1$ (from $x_1 = m_i \oplus R$), then B preprograms it into RO_2 by setting $T_2[K] \leftarrow m_i \oplus x_1$. To enforce that $K_{\nu(i)} = h_{i-1} \oplus x_2$ (from $x_2 = h_{i-1} \oplus K_{\nu(i)}$), algorithm B runs the mask reconstruction algorithm $\mathsf{MaskRec}$ of Appendix A on inputs $K, m_1, \ldots, m_i, x_2$ and obtains the $\mu_{\nu(1)}, \ldots, \mu_{\nu(\ell)}$ values. B programs these into RO_1 by setting $T_1[K, j] \leftarrow \mu_j$ for $0 \leq j \leq l$. The value obtained for $h_{i-1} \oplus K_{\nu(i)}$ then equals x_2. If any of the hash table entries $T_1[K, j]$ or $T_2[K]$ were already defined, then B aborts. This happens only when A was able to predict K though, which happens with probability $q_{\mathrm{RO}}/2^k$.

Algorithm B then runs A on input K and continues responding to its random oracle queries until A outputs its second preimage M'. Messages M and M' collide, if a collision occurs either in the final F call for $|M| \neq |M'|$, or for an internal F call in the $r\mathcal{SH}_{\mathrm{F}}$ chain for $|M| = |M'|$.

Whenever A wins, then B succeeds when it has embedded its challenge message at the correct i-th position ($i =$ correct) and A has asked not a valid random query (guessed the key) in the first phase of the game. Let E be the event that at least one of the preprogrammed masks is queried on a different input and let abort be the event that B aborts. Since B perfectly simulates A's environment, the advantage of B is given by

$$\begin{aligned}
\epsilon' &\geq \Pr[\,\mathsf{A}\text{ wins } \wedge\ i = \text{correct } \wedge\ \overline{\text{ABORT}}\,] \\
&= \Pr[\,\mathsf{A}\text{ wins } \wedge\ i = \text{correct } :\ \overline{\text{ABORT}}\,] \cdot \Pr[\,\overline{\text{ABORT}}\,] \\
&\geq \Pr[\,\mathsf{A}\text{ wins } \wedge\ i = \text{correct } :\ \overline{\text{ABORT}}\,] \cdot (1 - \Pr[\,\mathrm{E}\,]) \\
&\geq \frac{\epsilon}{\ell}\left(1 - \frac{q_{\mathrm{RO}}}{2^k}\right) \geq \frac{1}{\ell}\left(\epsilon - \frac{q_{\mathrm{RO}}}{2^k}\right) .
\end{aligned}$$

In estimate for the running time t, we assume that responding a random oracle query takes unit time. ▌

Theorem 5.3 If F is a (t', ϵ') Sec secure compression function, then $r\mathcal{SH}_{\mathrm{F}}$ is a $(t, q_{\mathrm{RO}}, \epsilon)$ Sec secure hash function in the random oracle model for $\epsilon \geq \ell\epsilon'$ and $t \leq t' - q_{\mathrm{RO}} - 2\ell \cdot \tau_{\mathrm{F}}$. Here, RO_1 and RO_2 are modeled as random oracles, τ_{F} is the time required for an evaluation of F, $\ell = \lceil \lambda / b \rceil$ and λ is the maximum message length.

Proof: (Sketch) The proof for Sec security preservation follows from the result of Theorem 5.2. Notice that here we get a tighter security bound because of the loss of $q_{\mathrm{RO}}/2^k$ term in the security reduction. This is true, because in the Sec preservation proof, B runs A after the masks R and $\mu_{\nu(1)}, \ldots, \mu_{\nu(\ell)}$ generation and their preprogramming in its hash tables $T_2[\cdot]$ and $T_1[\cdot]$ respectively. Hence A is not given access to a random oracle in the first mask generation phase. In the second phase, B runs A on the randomly chosen K and a random target message M. The answers to valid (for a valid key K) random oracle queries from A are valid existing entries from B's hash tables. ▌

Theorem 5.4 If F is a (t', ϵ') Pre secure compression function, then $r\mathcal{SH}_{\mathrm{F}}$ is a $(t, q_{\mathrm{RO}}, \epsilon)$ Pre secure hash function in the random oracle model for $\epsilon \geq \epsilon'$ and $t \leq t' - q_{\mathrm{RO}} - \ell \cdot \tau_{\mathrm{F}}$. Here, RO_1 and RO_2 are modeled as random oracles, τ_{F} is the time required for an evaluation of F, $\ell = \lceil \lambda / b \rceil$ and λ is the maximum message length.

Proof: (Sketch) The only difference with the proof of Theorem 3.2 is that instead of choosing random keys $R, K_0, \ldots, K_{l_{\max}}$, algorithm B chooses $K \xleftarrow{\$} \{0,1\}^k$ and lets the other keys be defined by the random oracles that are simulated in the usual way (similarly to the Proof of Theorem 5.1). ▌

5.2 Shorter Keys for the XOR-Tree

Given a compression function $\mathrm{F} : \{0,1\}^{an} \to \{0,1\}^n$ and random oracles $\mathrm{RO}_1 : \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^{an}$, $\mathrm{RO}_2 : \{0,1\}^k \to \{0,1\}^{an}$ where $l = \lceil \log_2 \log_2 \Lambda \rceil$, the *random-oracle XOR-Tree* is given by:

Algorithm $m\mathcal{XT}_{\mathrm{F}}(K, M)$:
  $m_1 \| \ldots \| m_{a^d} \leftarrow \mathrm{tpad}(M)$
  For $i = 1, \ldots, d$ do $K_i \leftarrow \mathrm{RO}_1(K, \langle i \rangle_l)$

$K^* \leftarrow \mathrm{RO}_2(K)$
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
$\quad h_{i,j} \leftarrow \mathrm{F}\left((h_{i-1,(j-1)a+1} \| \ldots \| h_{i-1,ja}) \oplus K_i\right)$
$h_{d+1,1} \leftarrow \mathrm{F}\left((h_{d,1} \| \langle |M| \rangle_{n(a-1)}) \oplus K^*\right)$
Return $h_{d+1,1}$

The tree padding function tpad is defined as for $m\mathcal{XT}_{\mathrm{F}}$. The $r\mathcal{TXOR}$ hash clearly builds on the $m\mathcal{XT}$ construction by generating all necessary keys using random oracles $\mathrm{RO}_1, \mathrm{RO}_2$. The theorems below show that collision, second-preimage, and preimage security are preserved in the $r\mathcal{TXOR}$ iteration. We present these theorems without proofs, simply because they are adaptations of the proofs for $m\mathcal{XT}$ in the same way that the proofs for $r\mathcal{SH}$ are adaptations of those for $\mathcal{SH}$.

Theorem 5.5 If there exists an explicitly given adversary A that $(t, q_{\mathrm{RO}}, \epsilon)$-breaks the Coll security of $r\mathcal{TXOR}_{\mathrm{F}}$ when $\mathrm{RO}_1, \mathrm{RO}_2$ are modeled as random oracles, then there exists an explicitly given adversary B that (t', ϵ')-breaks the Coll security of F for $\epsilon' \geq \epsilon$ and $t' \leq t + q_{\mathrm{RO}} + 2(\frac{a^d-1}{a-1} + 1) \cdot \tau_{\mathrm{F}}$, where τ_{F} is the time required for an evaluation of F, where d is the smallest integer such that $a^d \geq \lambda$, and λ is the maximum message length.

Theorem 5.6 If F is a (t', ϵ') Sec secure compression function, then $r\mathcal{TXOR}_{\mathrm{F}}$ is a $(t, q_{\mathrm{RO}}, \epsilon)$ eSec secure hash function in the random oracle model for $\epsilon \geq (\frac{a^d-1}{a-1}+1)\epsilon' + q_{\mathrm{RO}}/2^k$ and $t \leq t' - q_{\mathrm{RO}} - 2(\frac{a^d-1}{a-1}+1) \cdot \tau_{\mathrm{F}}$. Here, RO_1 and RO_2 are modeled as random oracles, τ_{F} is the time required for an evaluation of F, where d is the smallest integer such that $a^d \geq \lambda$, and λ is the maximum message length.

Theorem 5.7 If F is a (t', ϵ') Sec secure compression function, then $r\mathcal{TXOR}_{\mathrm{F}}$ is a $(t, q_{\mathrm{RO}}, \epsilon)$ Sec secure hash function in the random oracle model for $\epsilon \geq (\frac{a^d-1}{a-1} + 1)\epsilon'$ and $t \leq t' - q_{\mathrm{RO}} - 2(\frac{a^d-1}{a-1} + 1) \cdot \tau_{\mathrm{F}}$. Here, RO_1 and RO_2 are modeled as random oracles, τ_{F} is the time required for an evaluation of F, where d is the smallest integer such that $a^d \geq \lambda$, and λ is the maximum message length.

Theorem 5.8 If F is a (t', ϵ') Pre secure compression function, then $r\mathcal{TXOR}_{\mathrm{F}}$ is a $(t, q_{\mathrm{RO}}, \epsilon)$ Pre secure hash function in the random oracle model for $\epsilon \geq \epsilon'$ and $t \leq t' - q_{\mathrm{RO}} - (\frac{a^d-1}{a-1} + 1) \cdot \tau_{\mathrm{F}}$. Here, RO_1 and RO_2 are modeled as random oracles, τ_{F} is the time required for an evaluation of F, where d is the smallest integer such that $a^d \geq \lambda$, and λ is the maximum message length.

Acknowledgements

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 Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government and the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy). The work of the first author has been funded by a Ph.D. grant of the Flemish institute for BroadBand Technology (IBBT). The second author is a Postdoctoral Fellow of the Flemish Research Foundation (FWO - Vlaanderen). The fourth author was supported by NSF CNS-0627752.

References

[ANPS07] Elena Andreeva, Gregory Neven, Bart Preneel, and Thomas Shrimpton. Seven-property preserving iterated hashing: ROX. Cryptology ePrint Archive, Report, 2007. `http://eprint.iacr.org/`. (Cited on pages 2, 3, 4, 6, 8 and 13.)

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15, Santa Barbara, CA, USA, August 18–22, 1996. Springer-Verlag, Berlin, Germany. (Cited on page 3.)

[BR97] Mihir Bellare and Phillip Rogaway. Collision-resistant hashing: Towards making UOWHFs practical. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 470–484, Santa Barbara, CA, USA, August 17–21, 1997. Springer-Verlag, Berlin, Germany. (Cited on pages 1, 2, 5, 6, 8 and 14.)

[BR06] Mihir Bellare and Thomas Ristenpart. Multi-property-preserving hash domain extension: The EMD transform. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 299–314, Shanghai, China, December 3–7, 2006. Springer-Verlag, Berlin, Germany. (Cited on page 3.)

[CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448, Santa Barbara, CA, USA, August 14–18, 2005. Springer-Verlag, Berlin, Germany. (Cited on page 3.)

[Dam90] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427, Santa Barbara, CA, USA, August 20–24, 1990. Springer-Verlag, Berlin, Germany. (Cited on page 1.)

[KK06] John Kelsey and Tadayoshi Kohno. Herding hash functions and the Nostradamus attack. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200, St. Petersburg, Russia, May 28 – June 1, 2006. Springer-Verlag, Berlin, Germany. Available from `http://eprint.iacr.org/2005/281`. (Cited on page 3.)

[KS05] John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490, Aarhus, Denmark, May 22–26, 2005. Springer-Verlag, Berlin, Germany. (Cited on page 2.)

[Mer80] Ralph C. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Security and Privacy*, pages 122–134. IEEE Computer Society Press, 1980. (Cited on page 6.)

[Mer90] Ralph C. Merkle. One way hash functions and DES. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446, Santa Barbara, CA, USA, August 20–24, 1990. Springer-Verlag, Berlin, Germany. (Cited on page 1.)

[Mir01] Ilya Mironov. Hash functions: From Merkle-Damgård to Shoup. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 166–181, Innsbruck, Austria, May 6–10, 2001. Springer-Verlag, Berlin, Germany. (Cited on page 13.)

[Rog06] Phillip Rogaway. Formalizing human ignorance: Collision-resistant hashing without the keys. In *Vietcrypt 2006*, volume 4341 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2006. (Cited on pages 2, 3 and 4.)

[RS04] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer-Verlag, Berlin, Germany, 2004. (Cited on pages 1, 2, 3 and 4.)

[Sho00] Victor Shoup. A composition theorem for universal one-way hash functions. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 445–452, Bruges, Belgium, May 14–18, 2000. Springer-Verlag, Berlin, Germany. (Cited on pages 2, 4 and 5.)

[WY05] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35, Aarhus, Denmark, May 22–26, 2005. Springer-Verlag, Berlin, Germany. (Cited on page 2.)

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36, Santa Barbara, CA, USA, August 14–18, 2005. Springer-Verlag, Berlin, Germany. (Cited on page 2.)

A Shoup Key Schedule and Random Oracles

Algorithm $\mathsf{MaskRec}(m_1, \ldots, m_r, g)$
$j \leftarrow r$; For $i = 0, \ldots, t = \lfloor \log_2 r \rfloor$ do $\mu_i \leftarrow \perp$
Repeat while $j > 0$
$j' \leftarrow j - 2^{\nu(j)}$; $g' \xleftarrow{\$} \{0,1\}^n$
For $i = j'+1, \ldots, j-1$ do if $\mu_i = \perp$ then $\mu_i \xleftarrow{\$} \{0,1\}^n$
If $j' = 0$ then $h_0 \leftarrow IV$ else $h_{j'} \leftarrow \mathrm{F}(m_{j'} \| g')$
For $i = j'+1, \ldots, j-1$ do
$g_i \leftarrow h_{i-1} \oplus \mu_{\nu(i)}$; $h_i \leftarrow \mathrm{F}(m_i \| g_i)$
$\mu_{\nu(j)} \leftarrow g_{j-1} \oplus g'$; $j \leftarrow j'$; $g \leftarrow g'$
For $i = 0, \ldots, t$ do if $\mu_i = \perp$ then $\mu_i \xleftarrow{\$} \{0,1\}^n$
Return $(\mu_0, \ldots, \mu_t)$

Algorithm $\mathsf{RO\text{-}Sim}_1(K, x)$
If $T_1[K, x] = \perp$ then
$T_1[K, x] \xleftarrow{\$} \{0,1\}^n$
Return $T_1[K, x]$

Algorithm $\mathsf{RO\text{-}Sim}_2(K)$
If $T_2[K] = \perp$ then
$T_2[K] \xleftarrow{\$} \{0,1\}^b$
Return $T_2[K]$

Figure 3: **Algorithms used in the proofs.** On the left, we recall the mask reconstruction algorithm of [Mir01] (M is parsed as b-bit blocks). On the right, we give the algorithms used to simulate the random oracles.

B Merkle-Damgård does not Preserve (Second) Preimage Resistance

We show that the strengthened Merkle-Damgård ($s\mathcal{MD}$) hash does not preserve the Sec and Pre-security by building the following Sec and Pre-secure counterexample compression function.

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

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

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

If G is (t, ϵ) atk secure, then CE_1 is $(t, \epsilon - 1/2^n)$ atk secure. We refer to [ANPS07] for the proof. From the construction of CE_1, it is clear that $s\mathcal{MD}_{\mathrm{CE}_1}(M) = IV$ for all $M \in \{0,1\}^*$. Hence, the adversary can output any message M' as its (second) preimage. ∎

C Preimage and Collision Resistance of XOR-Tree

In this section we provide the Pre-security preservation and Coll-security proofs for XOR-Tree ($\mathcal{XT}$) hash. We could neither prove or disprove its Coll or Sec preservation. Below we recall the XOR-Tree algorithm of [BR97] for variable length messages.

Algorithm $\mathcal{XT}_{\mathrm{F}}(K_1\| \ldots \|K_{d+1}\ ,\ M)$:
$m_1\| \ldots \|m_\ell \leftarrow \mathrm{tpad}(M)$
For $i = 1, \ldots, a^{d-1}$ 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 $h_{i,j} \leftarrow \mathrm{F}((h_{i-1,(j-1)a+1}\| \ldots \|h_{i-1,ja}) \oplus K_i)$
$h_{d+1,1} \leftarrow \mathrm{F}((h_{d,1}\|\langle |M| \rangle_{n(a-1)}) \oplus K_{d+1})$
Return $h_{d+1,1}$.

First we provide the Pre-security preservation proof of the $\mathcal{XT}_{\mathrm{F}}$ iteration.

Theorem C.1 If F is (t', ϵ') Pre-secure, then $\mathcal{XT}_{\mathrm{F}}$ is (t, ϵ) Pre-secure for $\epsilon \geq \epsilon'$ and $t \leq t' - (\frac{a^d-1}{a-1} + 1) \cdot \tau_{\mathrm{F}}$, where τ_{F} is the time needed for an evaluation of F, where λ is the length of the message output by A, and where d is the smallest integer such that $a^d \geq \lambda$.

Proof: Given an Pre[λ]-adversary A against $\mathcal{XT}_{\mathrm{F}}$, consider the following Pre-adversary B against F. B obtains a target value Y (computed over a random message input $x = x_1\| \ldots \|x_a$ of an-bits). B generates at random $an(d+1)$-bit keys $K_1\| \ldots \|K_{d+1}$. B runs A on the same target value Y and generated keys to obtain A's preimage message M'. Let $m'_1\| \ldots \|m'_\ell \leftarrow \mathrm{tpad}(M')$ and let $h'_{d,1}$ be the one-but-last output hash value computed in the execution of $\mathcal{XT}_{\mathrm{F}}(K_1\| \ldots \|K_{d+1}, M')$. Algorithm B outputs $(h'_{d,1}\|\langle |M| \rangle_{(a-1)n}) \oplus K_{d+1}$ as its own preimage. The running time of B equals that of A plus up to $2(\frac{a^d-1}{a-1} + 1)$ compression function evaluations. Note that the number of compression function calls (nodes) in the $m\mathcal{XT}$ tree equals $\left(\sum_{i=0}^{d-1} a^i\right) + 1 = \frac{a^d-1}{a-1} + 1$. ∎

Next we prove the Coll-security of the XOR-Tree hash under the two following assumptions: 1) Coll-security of the underlying compression function F; 2) δ-Coll-security of F. For an adversary A attacking the compression function F we define the δ-Coll advantage measure as:

$$\mathbf{Adv}_{\mathrm{F}}^{\delta\text{-Coll}}(\mathsf{A}) \quad = \quad \Pr\left[\ \delta \xleftarrow{\$} \{0,1\}^n\ ;\ M', M \xleftarrow{\$} \mathsf{A}(\delta) \quad : \quad M \neq M' \text{ and } \mathrm{F}(M) \oplus \mathrm{F}(M') = \delta\ \right]$$

The δ-Coll security notions covers the intuition that no efficient adversary A should be able to output two messages that have a randomly fixed XOR hash difference under F. We clarify, that the δ-Coll and Coll-security properties are independent from each other. There exist contrived counterexamples which show that Coll does not imply δ-Coll and vice versa.

Theorem C.2 If there exists an explicitly given adversary A that (t, ϵ)-breaks the Coll security of $m\mathcal{XT}_{\mathrm{F}}$, then there exists explicitly given adversaries B_1 that (t', ϵ')-breaks the Coll security of F and B_2 that (t'', ϵ'')-breaks the δ-Coll security of F for $\epsilon \leq \epsilon' + \frac{2\epsilon''}{(\log_a \ell)^2}$ and $t' + t'' \cdot \tau_{\mathrm{F}} \leq t + 4(\frac{a^d-1}{a-1} + 1)$, where τ_{F} is the time needed for one evaluation of F, d is the smallest integer such that $a^d \geq \lambda$, and λ is the maximum message length.

Proof: Given a Coll-adversary A against $\mathcal{XT}_{\mathrm{F}}$, we construct a Coll adversary B_1 and a δ-Coll adversary B_2 against F.

Let us consider first B_1 that generates at random the $n(d+1)$ -bit sequence of key strings $K_1\| \ldots \|K_{d+1}$. B_1 runs A on the same key sequence to obtain a pair of colliding messages M and M'. Let $m_1\| \ldots \|m_\ell \leftarrow$

$\mathrm{tpad}(M)$, $m'_1\| \ldots \|m'_{\ell'} \leftarrow \mathrm{tpad}(M')$, and let $h_{d,1}$ and $h'_{d',1}$ be the one-but-last output hash values computed in the execution of $\mathcal{XT}_{\mathrm{F}}$ on inputs M and M' respectively.

In case $|M| = |M'|$ $(d = d')$ and $(h_{d,1}\|\langle |M| \rangle_{(a-1)n}) \oplus K_{d+1} \neq (h'_{d,1}\|\langle |M'| \rangle_{(a-1)n}) \oplus K_{d+1}$ then B_1 outputs $(h_{d,1}\|\langle |M| \rangle_{(a-1)n}) \oplus K_{d+1}$ and $(h'_{d,1}\|\langle |M'| \rangle_{(a-1)n}) \oplus K_{d+1}$ as its valid colliding messages. If these two values are equal, then B_1 goes one level up to compare the inputs to F. If they are different, then B_1 outputs these as its colliding pair. Else, B_1 continues as described until it reaches the top tree level. Unless $M = M'$, there always exists an index $i \in \{1, a+1, 2a+1, \ldots, a(a^{\log_a \ell - 1} - 1) + 1\}$, such that at least one pair of message inputs $m_i\| \ldots \|m_{ia}$ and $m'_i\| \ldots \|m'_{ia}$ differs and forms a valid colliding pair for B_1.

When $|M| \neq |M'|$ $(d \neq d')$, then if $(h_{d,1}\|\langle |M| \rangle_{(a-1)n}) \oplus K_{d+1} \neq (h'_{d',1}\|\langle |M'| \rangle_{(a-1)n}) \oplus K_{d'+1}$, B_1 outputs $(h_{d,1}\|\langle |M| \rangle_{(a-1)n}) \oplus K_{d+1}$ and $(h'_{d',1}\|\langle |M'| \rangle_{(a-1)n}) \oplus K_{d'+1}$ as its valid colliding messages. B_1 can not continue further when $(h_{d,1}\|\langle |M| \rangle_{(a-1)n}) \oplus K_{d+1} = (h'_{d',1}\|\langle |M'| \rangle_{(a-1)n}) \oplus K_{d'+1}$. In that case B_1 aborts.

Given the Coll adversary A against $\mathcal{XT}_{\mathrm{F}}$, we proceed by building a new adversary B_2 against the δ-Coll-security of F. We show that B_2 succeeds in these cases in which B_1 aborts. B_2 obtains a random n-bit string δ and chooses $\delta' \xleftarrow{\$} \{0,1\}^{(a-1)n}$ at random. B_2 selects two indexes i and j, such that $i \xleftarrow{\$} \{1, \ldots, d+1\}$, and $j \xleftarrow{\$} \{1, \ldots, d_{\max}+1\} \backslash i$ ($d_{\max}$ is the maximal tree depth). B_2 sets K_i, such that $K_i = (\delta\|\delta') \oplus K_j$ and generates and a random nd-bit sequence of key strings $K_1\| \ldots \|K_{i-1}$ and $K_{i+1}\| \ldots \|K_{d_{\max}}$. Then B_2 runs A on input the generated key sequence $K_1 \ldots K_{d_{\max}+1}$ to obtain colliding messages M and M' from A.

If $|M| = |M'|$, then B_2 aborts (note that B_1 does always succeed in this case). Else if $|M| \neq |M'|$ and $(h_{d,1}\|\langle |M| \rangle_{(a-1)n}) \oplus K_{d+1} = (h'_{d',1}\|\langle |M'| \rangle_{(a-1)n}) \oplus K_{d'+1}$, then B_2 has successfully embedded a valid key difference for $K_{d+1} = (\delta\|\delta') \oplus K_{d'+1}$ ($i = d+1$ and $j = d'+1$, or vice versa). In this case B_2 outputs $(h_{d-1,1}\| \ldots h_{d-1,(a-1)n}) \oplus K_d$ and $(h'_{d'-1,1}\| \ldots h'_{d'-1,(a-1)n}) \oplus K_{d'}$ as its δ-Coll colliding pair of messages. (Note that $\mathrm{F}((h_{d-1,1}\| \ldots h_{d-1,(a-1)n}) \oplus K_d) \oplus \mathrm{F}((h'_{d'-1,1}\| \ldots h'_{d'-1,(a-1)n}) \oplus K_{d'})$ equals exactly δ). B_2 succeeds only when the randomly selected indexes i and j equal $d+1$ and $d'+1$, else it aborts. We denote this event by E and show that the probability of event E to occur is

$$\Pr[\,\mathrm{E}\,] \;=\; \Pr\big[\,\big(i = d+1 \wedge j = d'+1\big) \vee \big(j = d+1 \wedge i = d'+1\big)\,\big] \leq \frac{2}{(\log_a \ell)^2}\ .$$

If $(h_{d,1}\|\langle |M| \rangle_{(a-1)n}) \oplus K_{d+1} \neq (h'_{d',1}\|\langle |M'| \rangle_{(a-1)n}) \oplus K_{d'+1}$, then B_2 aborts (remember that in this case B_1 is successful).

We can express the advantage of A by

$$\begin{aligned}
\epsilon &\leq \Pr[\,\mathsf{B}_1 \text{ wins } \vee\ (\mathsf{B}_2 \text{ wins } \wedge\ \mathrm{E})\,] \\
&= \Pr[\,\mathsf{B}_1 \text{ wins}\,] + \Pr[\,\mathsf{B}_2 \text{ wins } :\ \mathrm{E}\,] \cdot \Pr[\,\mathrm{E}\,] \\
&\leq \epsilon' + \Pr[\,\mathsf{B}_2 \text{ wins } :\ \mathrm{E}\,] \cdot \frac{2}{(\log_a \ell)^2} \leq \epsilon' + \frac{2\epsilon''}{(\log_a \ell)^2}\ .
\end{aligned}$$

The running times of B_1 and B_2 equal that of A plus at most 4ℓ evaluations of F. ▌