Hash Functions in the Dedicated-Key Setting: Design Choices and MPP Transforms

Mihir Bellare and Thomas Ristenpart

Dept. of Computer Science & Engineering 0404, University of California San Diego
9500 Gilman Drive, La Jolla, CA 92093-0404, USA
{mihir,tristenp}@cs.ucsd.edu
http://www-cse.ucsd.edu/users/{mihir,tristenp}

Abstract. In the dedicated-key setting, one uses a compression function $f\colon \{0,1\}^k \times \{0,1\}^{n+d} \to \{0,1\}^n$ to build a family of hash functions $H^f\colon \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$ indexed by a key space $\mathcal{K}$. This is different from the more traditional design approach used to build hash functions such as MD5 or SHA-1, in which compression functions and hash functions do not have dedicated key inputs. We explore the benefits and drawbacks of building hash functions in the dedicated-key setting (as compared to the more traditional approach), highlighting several unique features of the former. Should one choose to build hash functions in the dedicated-key setting, we suggest utilizing multi-property-preserving (MPP) domain extension transforms. We analyze seven existing dedicated-key transforms with regard to the MPP goal and propose two simple new MPP transforms.

1 Introduction

Two settings. A popular method for designing hash functions proceeds as follows. First, one designs a compression function $f\colon \{0,1\}^{d+n} \to \{0,1\}^n$, where d is the length of a data block and n is the length of the chaining variable. Then one specifies a *domain extension transform* H that utilizes f as a black box to implement the hash function $H^f\colon \mathcal{M} \to \{0,1\}^n$ associated to f, where $\mathcal{M}$ is some large message space. Most in-use hash functions, for example the MD-x family [21] and SHA-1 [19], were constructed using this approach.

There also exists a second setting for hash function design and analysis, in which compression functions and hash functions both have a *dedicated key input*. A dedicated-key compression function has signature $f\colon \{0,1\}^k \times \{0,1\}^{d+n} \to \{0,1\}^n$. A transform H now uses $f(\cdot,\cdot)$ as a black-box to implement a family of hash functions $H^f\colon \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$ indexed by a key space $\mathcal{K}$. We call this the *dedicated-key setting*. Note that although we use the term "key", this does not mean that a key $K \in \mathcal{K}$ is necessarily private. Indeed, hash functions often *need* to be publicly computable (e.g., for verifying digital signatures) and so in these settings every party must have access to the key.

This paper. Due to recent collision-finding attacks against in-use hash functions such as MD5 and SHA-1 [26,27], new hash functions are going to be designed

L. Arge et al. (Eds.): ICALP 2007, LNCS 4596, pp. 399–410, 2007.

and standardized. A crucial choice for designers will be whether one should build hash functions in the first setting (continuing in the current tradition of in-use hash functions) or in the dedicated-key setting. Our first contribution is to present relative merits of the two settings described above, pointing out several important benefits of the dedicated-key setting, but also its most significant drawbacks.

Should one choose to work in the dedicated-key setting, the natural next question is how to best build hash functions in it. Because hash functions are currently used in a wide variety of applications with disjoint security requirements, we suggest building hash functions using *multi-property-preserving* (MPP) transforms, introduced for the non-dedicated-key setting in [6]. An MPP transform H simultaneously *preserves* numerous properties of interest: if the compression function f has security property P, then H^f has P also. Our second contribution is an MPP-orientated analysis of several dedicated-key transforms and the proposal of two new transforms that better meet the MPP goal.

We now briefly summarize our results in more detail.

The dedicated-key setting. In Section 3, we discuss the dedicated-key setting, pointing out several features which are distinct from the more traditional setting. We describe two important benefits of the dedicated-key setting: hash function heterogeneity (allowing users to specify independent instances of the hash function) and improved security guarantees (particularly for message authentication, a wide-spread application of hash functions). On the other hand, a significant downside of dedicated keys is a decrease in efficiency.

Dedicated-key transforms. In Section 5, we provide an MPP-orientated treatment of transforms in the dedicated-key setting, analyzing seven previously proposed Merkle-Damgård-like transforms: plain Merkle-Damgård (MD) [16,12], strengthened MD (sMD) [12], prefix-free MD (Pre) [15], Shoup's transform (Sh) [24], the strengthened Nested Iteration transform (sNI) [1], the Nested Iteration transform (NI) [15], and the Chain-Shift transform (CS) [15]. Figure 1 summarizes our results for the existing seven transforms. For each transform we determine if it is collision-resistance preserving (CR-Pr), message authentication code preserving (MAC-Pr), pseudorandom function preserving (PRF-Pr), and pseudorandom oracle preserving (PRO-Pr). A "Yes" in the P-Pr column for transform T means that, if a compression function f has property P, then T^f provably has property P. A "No" means that there exists a compression function f with property P, but for which T^f does *not* have P. Only one of the seven transforms preserves the first four properties (though requiring two keys to do so), and so we suggest a new MPP transform, called Strengthened Chain-Shift, which is efficient and requires just one key.

We also investigate the property of being a universal one-way hash function [18], which we'll call target-collision resistance (following [10]). The practical value of TCR-Pr transforms is limited by the significant amount of key material they require; see Section 5.5 for a discussion. That said, none of the transforms thus far preserve it along with the other four properties, and so we suggest a new transform, Enveloped Shoup, which preserves all five properties.

	CR-Pr	MAC-Pr	PRF-Pr	PRO-Pr	TCR-Pr	Efficiency $\tau(L)$	Key bits
MD	No [16,12]	**No**	**Yes**	No [11]	No [10]	$\lceil (L+1)/d \rceil$	k
sMD	Yes [16,12]	**No**	**Yes**	No [11]	No [10]	$\lceil (L+65)/d \rceil$	k
Pre	**No**	Yes [15]	**Yes**	Yes [11]	**No**	$\lceil (L+1)/(d-1) \rceil$	k
Sh	Yes [24]	**No**	**Yes**	**No**	Yes [24]	$\lceil (L+65)/d \rceil$	$k + n\lceil \log_2 \tau(L) \rceil$
sNI	**Yes**	Yes [1]	**Yes**	Yes [6]	**No**	$\lceil (L+65)/d \rceil$	$2k$
NI	**No**	Yes [15]	**Yes**	Yes [6]	**No**	$\lceil (L+1)/d \rceil$	$2k$
CS	**No**	Yes [15]	**Yes**	Yes [6]	**No**	$\lceil (L+1+n)/d \rceil$	k
sCS	**Yes**	Yes [15]	**Yes**	Yes [6]	**No**	$\lceil (L+65+n)/d \rceil$	k
ESh	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	$\lceil (L+65+n)/d \rceil$	$k + n\lceil \log_2 \tau(L) \rceil$

Fig. 1. Summary of transforms in the dedicated-key setting when applied to a compression function $f\colon \{0,1\}^k \times \{0,1\}^{n+d} \to \{0,1\}^n$. Bold-faced claims are novel. Efficiency is measured by $\tau(L)$, the number of compression function applications used to hash an L-bit string.

2 Notation and Definitions

NOTATION. We denote pairwise concatenation by $||$, e.g. $M \,||\, M'$, and write $M_1 \cdots M_k$ to mean $M_1 \,||\, M_2 \,||\, \ldots \,||\, M_k$. For brevity, we define the following semantics for the notation $M_1 \cdots M_k \xleftarrow{d} M$ where M is a string of bits: 1) define $k = \lceil |M|/d \rceil$ and 2) if $|M| \bmod d = 0$ then parse M into $M_1, M_2, \ldots, M_k$ where $|M_i| = d$ for $1 \le i \le k$, otherwise parse M into $M_1, M_2, \ldots, M_{k-1}, M_k$ where $|M_i| = d$ for $1 \le i \le k-1$ and $|M_k| = |M| \bmod d$. For any finite set S we write $s \xleftarrow{\$} S$ to signify uniformly choosing a value $s \in S$. A random oracle is an algorithm $\mathsf{RF}_{Dom,Rng}$ that, on input $X \in Dom$, returns a value $Y \xleftarrow{\$} Rng$. Repeat queries are, however, answered consistently. We sometimes write $\mathsf{RF}_{d,r}$ when $Dom = \{0,1\}^d$ and $Rng = \{0,1\}^r$.

SECURITY NOTIONS. Let $F\colon \mathcal{K} \times Dom \to Rng$ be a function with non-empty key space $\mathcal{K}$ and define $F_K(\cdot) = F(K,\cdot)$. Then we define the following security experiments:

- tcr: $\epsilon = \Pr\left[(X,S) \xleftarrow{\$} \mathcal{A}_1, K \xleftarrow{\$} \mathcal{K}, X' \xleftarrow{\$} \mathcal{A}_2(S,K) : \begin{array}{c} X \neq X' \wedge \\ F_K(X) = F_K(X') \end{array} \right]$
- cr: $\epsilon = \Pr\left[K \xleftarrow{\$} \mathcal{K}, (X,X') \xleftarrow{\$} \mathcal{A}(K) : X \neq X' \wedge F_K(X) = F_K(X') \right]$
- mac: $\epsilon = \Pr\left[K \xleftarrow{\$} \mathcal{K}, (X,T) \xleftarrow{\$} \mathcal{A}^{F(K,\cdot)} : F_K(X) = T \wedge X \text{ not queried} \right]$
- prf: $\epsilon = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{F(K,\cdot)} \Rightarrow 1 \right] - \Pr\left[\rho \xleftarrow{\$} \mathrm{Func}(Dom, Rng) : \mathcal{A}^{\rho(\cdot)} \Rightarrow 1 \right]$

where the probabilities are over the specified random choices and the coins used by $\mathcal{A}$. In the tcr game $\mathcal{A}$ is actually a pair of algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$. Now letting F be an algorithm given oracle access to an ideal compression function $f = \mathsf{RF}_{k+n+d,n}$ we define the last security experiment:

- pro: $\epsilon = \Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{F_K^f, f}(K) \Rightarrow 1 \right] - \Pr\left[K \xleftarrow{\$} \mathcal{K} : A^{\mathcal{F}, \mathcal{S}_K^{\mathcal{F}}}(K) \Rightarrow 1 \right]$

where the probabilities are over the specified random choices, the coins used by $\mathcal{A}$ and $\mathcal{S}$, and the coins used by $\mathcal{F} = \mathsf{RF}_{Dom,Rng}$ and $f = \mathsf{RF}_{n+d,n}$. The simulator $\mathcal{S}$ maintains state across queries and has oracle access to $\mathcal{F}$. For more details on the pseudorandom oracle definition see [6,11,13].

We say that F is (t, L, ϵ)-xxx for xxx $\in \{\text{tcr}, \text{cr}\}$ if any adversary $\mathcal{A}$ running in time at most t and outputting messages of length less than or equal to L bits has at most ϵ probability of success in the xxx game. Similarly we say that F is a (t, q, L, ϵ)-xxx for xxx $\in \{\text{mac}, \text{prf}\}$ if any adversary $\mathcal{A}$ running in time at most t and making at most q queries each of which has length at most L has at most ϵ probability of success in the xxx game. Lastly we say that F is a $(t_{\mathcal{A}}, t_{\mathcal{S}}, q_1, q_2, L, \epsilon)$-pro if there exists a simulator $\mathcal{S}$ running in time $t_{\mathcal{S}}$ such that the following is true. Any adversary $\mathcal{A}$ running in time at most $t_{\mathcal{A}}$ and asking at most q_1 (q_2) queries to its first (second) oracle, with maximal query length L bits, has probability at most ϵ of success in the pro game.

3 Hash Functions in the Dedicated Key Setting

Hash function heterogeneity. The first major benefit of dedicated-key hash functions is the enablement of *hash function heterogeneity*, which allows for the utilization of numerous different hash function instances. To understand why this is useful for security, we discuss (as an example) an important application of publicly-computable, collision-resistant hash functions: digital signature schemes. Recall that in such a scheme each party i picks a public key pk_i and publishes it. To verify a message, one hashes it and then applies some verification algorithm that utilizes pk_i. In current practice, all users utilize a single hash function H^h, for example SHA-1. Now that Wang, Yin, and Yu discovered a collision-outputting algorithm $\mathcal{A}$ against $H^h = \text{SHA-1}$ [26], simply running $\mathcal{A}$ a single time compromises the security of *every* user's digital signature scheme.

If we instead utilize a dedicated-key hash function $H^h\colon \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$ within our scheme, then each user i can pick a key $K_i \in \mathcal{K}$ and publish it as part of their public key. In this way each user has his or her own hash function instance, exemplifying hash function heterogeneity. Now, attackers are faced with a significantly more difficult task, from a practical perspective. If they can construct a *categorical* attack algorithm $\mathcal{A}$ (i.e., one that works equally well on any key), and if $\mathcal{A}$ executes in w operations, then to attack a single user i requires (as before) w work. But attacking two users requires $2w$ work, and in general attacking a group of p users requires pw work. If $w \approx 2^{69}$, as is the case for Wang, Yin, and Yu's SHA-1 attack [26], then even doubling the amount of work is a significant hurdle to mounting attacks in practice. The situation is even worse for the attackers if their attack algorithm is *key-specific* (i.e., it only works well on a particular key), because then they might have to adapt their attack to each user's key, which could require more cryptanalytic effort. In either case, hash function heterogeneity is a significant benefit of the dedicated-key setting, particularly when attacks are found that are just on the cusp of practicality.

Improved security guarantees. An important and wide-spread application of hash functions is for message authentication code (MAC) schemes, where we require hash functions to be unforgeable. To utilize a traditional hash function $H^h\colon \mathcal{M} \to \{0,1\}^n$ as a MAC scheme, H^h must be *keyed*, which means some of the input is set aside (a posteriori) for key bits. The canonical construct in this domain is HMAC [3,2], which is widely standardized and used. (NIST FIPS 198, ANSI X9.71, IETF RFC 2104, SSL, SSH, IPSEC, TLS, IEEE 802.11i, and IEEE 802.16e are only some instances.) Note that in these applications keys are secret and never revealed publicly.

In the traditional setting, the unforgeability of MACs built from hash functions requires the compression function to be a pseudorandom function (PRF), when keyed appropriately, and the transform to be PRF-Pr. However, unforgeability is a *weaker* security goal than being a PRF: any PRF is a good MAC but not vice versa. The reason we have to rely on PRFs for message authentication is that building transforms that preserve the unforgeability of a compression function $h\colon \{0,1\}^{n+d} \to \{0,1\}^n$ is inherently difficult and, in fact, no unforgeability preserving (which we'll call MAC-Pr) transforms are known in this setting.

On the other hand, if we work in the dedicated-key setting, then there are straightforward MAC-Pr transforms [1,15,14]. This allows us to utilize hash functions as MACs under just the assumption that $h\colon \{0,1\}^k \times \{0,1\}^{n+d} \to \{0,1\}^n$ is a good MAC, which provides a better security guarantee. To see why, note that an attack showing that h is not a PRF does *not* immediately imply that h can be forged against and therefore we can still have a guarantee that H^h is a secure MAC — but this is only true in the dedicated-key setting. In the prior setting we would lose all security guarantees.

Another benefit of the dedicated-key setting is that building transforms which are provably PRF-Pr becomes much easier. As we show in Section 5.3, *all* the transforms we consider are PRF-Pr, and the proofs of this are straightforward.

Keying and collision-resistance. Hash functions with dedicated key inputs are an easy solution for the *foundations-of-hashing dilemma* [22], which is a problem of theoretical interest. The dilemma refers to the fact that $h\colon \{0,1\}^{n+d} \to \{0,1\}^n$, for $d \neq 0$, can *not* be collision-resistant: by the pigeonhole principle there are two distinct strings $X, X' \in \{0,1\}^{n+d}$ such that $h(X) = h(X')$. Thus there always exists an efficient collision-outputting algorithm $\mathcal{A}$, namely the one that outputs (X, X'). However, as Rogaway discusses at length in [22], rigorous provable security for keyless hash functions is still meaningful, since we can give explicit reductions (though at the cost of slightly more complex theorem statements). So while the dedicated-key setting enables formally meaningful collision-resistance and thus simpler theoretical treatments of CR hashing, the practical impact of this benefit is small.

Efficiency. A significant downside of dedicated keys is efficiency loss. For every message block hashed using a dedicated-key compression function $h\colon \{0,1\}^k \times \{0,1\}^{n+d} \to \{0,1\}^n$, a total of $k+n+d$ bits must be processed. Compare this to the situation of current popular hash functions, which only have to process $n+d$ bits per block. The efficiency of the hash function therefore goes down by

about $\frac{k}{n+d}$, which could be an issue in settings where speed is paramount (e.g., message authentication of network traffic).

Backwards-compatibility. In many settings it will be desirable to utilize a hash function that does *not* reveal a dedicated-key input. This will be particularly true for backwards-compatibility with existing applications that utilize a standard hash function $H\colon \{0,1\}^* \to \{0,1\}^n$. We point out that it is easy to allow such compatibility when starting with a dedicated-key hash function $H'\colon \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n$. Simply fix an honestly generated and publicly-known key K (chosen, for example, by some trusted entity), and define the unkeyed hash function as $H(M) = H'(K, M)$.

Adversarially-chosen keys. Most current cryptographically-sanctified applications of hash functions (e.g., digital signature schemes, message authentication codes, key derivation, and standard uses of random oracles) only require security for honestly generated dedicated keys. However, given the wide-spread use of hash functions in non-standard settings, one should be aware of the potential for abuse in applications that would require security even in the face of adversarially-chosen keys. A simple solution for such settings could be to require a fixed, honestly-generated key as mentioned above.

4 Dedicated Key Transforms

Let $f\colon \{0,1\}^k \times \{0,1\}^{n+d} \to \{0,1\}^n$ be a dedicated-key compression function with $d \geq n \geq 64$. We now describe the various transforms treated in this paper. A transform H describes how to utilize f (as a black box) in order to generate a hash function $H^f\colon \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$. A transform is defined in two separate steps. We first specify an injective padding function that maps from $\{0,1\}^*$ or $\{0,1\}^{\leq 2^{64}}$ to either $D^+ = \cup_{i \geq 1}\{0,1\}^{id}$ or $D^\circ = \cup_{i \geq 1}\{0,1\}^{id+d-n}$. Then we specify an iteration function which describes how to hash strings in either D^+ or D°. We define the following padding functions:

- $\mathsf{pad}\colon \{0,1\}^* \to D^+$ is defined by $\mathsf{pad}(M) = M \,||\, 10^r$
- $\mathsf{pad_s}\colon \mathcal{D} \to D^+$ is defined by $\mathsf{pad_s}(M) = M \,||\, 10^r \,||\, \langle |M| \rangle_{64}$
- $\mathsf{padPF}\colon \{0,1\}^* \to D^+$ is a prefix-free padding function: for any $M, M' \in \{0,1\}^*$ where $|M| < |M'|$ we have that $\mathsf{padPF}(M)$ is not a prefix of $\mathsf{padPF}(M')$. (For example: pad to make the length a multiple of $d-1$, parse the result into blocks of $d-1$ bits, append a zero to each block except the final block, and append one to the final block.)
- $\mathsf{padCS}\colon \{0,1\}^* \to D^\circ$ is defined by $\mathsf{padCS}(M) = M \,||\, 10^r$
- $\mathsf{padCS_s}\colon \mathcal{D} \to D^\circ$ is defined by $\mathsf{padCS_s}(M) = M \,||\, 10^r \,||\, \langle |M| \rangle_{64} \,||\, 0^p$

where for pad, $\mathsf{pad_s}$, and padCS the value r is the minimal number of zeros so that the returned string is in the range of the padding function. For $\mathsf{padCS_s}$ we define p in two potential ways. If $d \geq n + 64$ (there is room for the strengthening in the envelope), then $p = 0$. If $d < n + 64$ (there is not enough room for the strengthening in the envelope), then $p = d - n$. Then r is the number of zeros

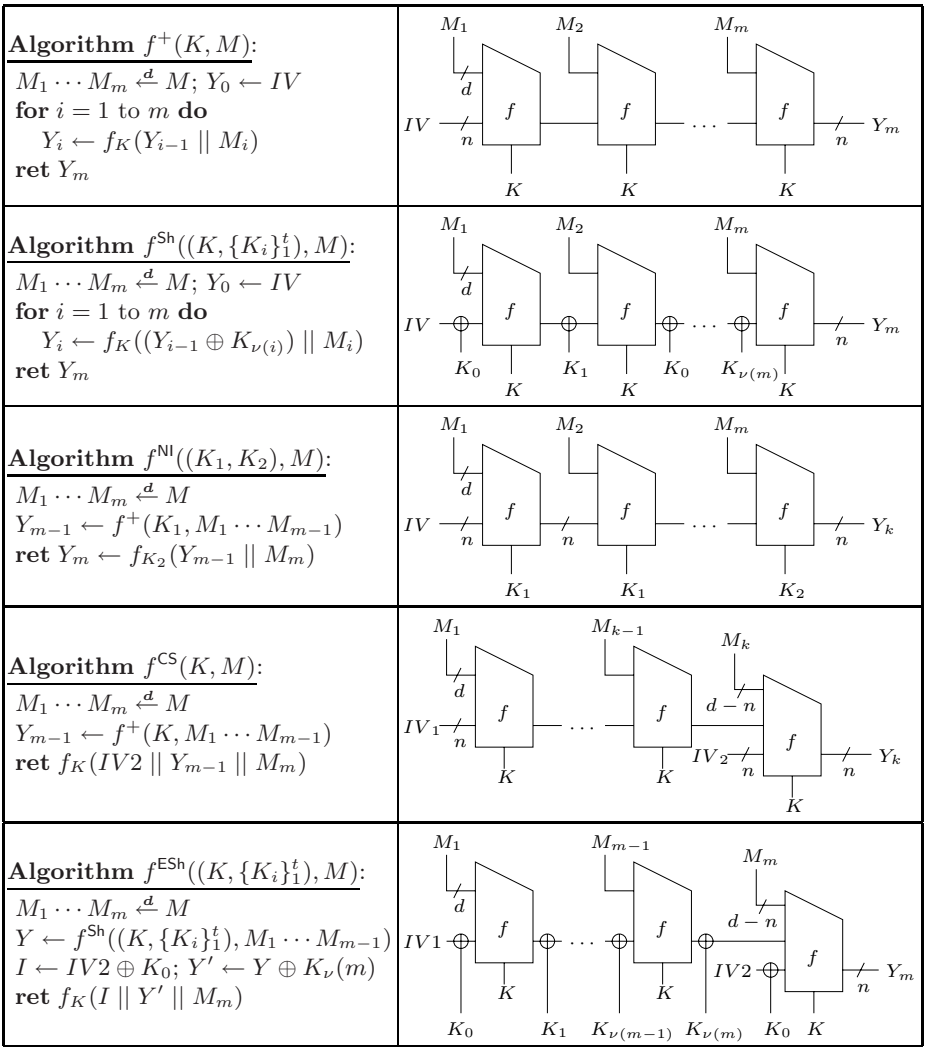

Fig. 2. The algorithms and diagrams detailing the iteration functions we consider. Transforms are the composition of an iteration function and a padding function.

needed to make the returned string in D°. Note that we restrict our attention to padding functions g such that for any messages M, M' for which $|M| = |M'|$ we have that $|g(M)| = |g(M')|$.

The iteration functions we consider are specified in Figure 2, and we use them to now define the seven previously proposed transforms. The basic Merkle-Damgård (MD) iteration f^+: $\{0,1\}^k \times D^+ \to \{0,1\}^n$ repeatedly applies f. The **plain MD** transform [16,12] is $\mathsf{MD}[f] = f^+(\mathrm{K}, \mathsf{pad}(\mathrm{M}))$. The **strengthened MD** transform [12] is $\mathsf{sMD}[f] = f^+(\mathrm{K}, \mathsf{pad_s}(\mathrm{M}))$. The **prefix-free MD**

transform [15] is $\mathsf{Pre}[f] = f^{+}(\text{K}, \mathsf{padPF}(\text{M}))$. The placeholders K and M designate how to handle the key and message inputs. The Shoup iteration f^{Sh}: $(\{0,1\}^k \times \{0,1\}^{tn}) \times D^{+} \to \{0,1\}^n$ utilizes $t = \lceil \log_2(\sigma) \rceil$ key masks where σ the maximal number of iterations of f allowed. Also we define $\nu(i)$ to be the largest value x such that 2^x divides i. The key masks $\{K_i\}_1^t \in \{0,1\}^{tn}$ are t n-bit keys that are used to 'mask' chaining variables with secret key material. Then the **Shoup** transform [24] is $\mathsf{Sh}[f] = f^{\mathsf{Sh}}(\text{K}, \mathsf{pad_s}(\text{M}))$. The nested iteration f^{NI}: $(\{0,1\}^k \times \{0,1\}^k) \times D^{+} \to \{0,1\}^n$ just envelopes an f^{+} iteration with an application of f using a second key. The **strengthened Nested Iteration** transform [1] is $\mathsf{sNI}[f] = f^{\mathsf{NI}}(\text{K}, \mathsf{pad_s}(\text{M}))$. The **Nested Iteration** transform [15] is $\mathsf{NI}[f] = f^{\mathsf{NI}}(\text{K}, \mathsf{pad}(\text{M}))$. The chain shift iteration f^{CS}: $\{0,1\}^k \times D^{\circ} \to \{0,1\}^n$, envelopes an internal f^{+} iteration with an application of $f(IV2 \,||\, \cdot)$. We require that $IV2 \neq IV1$. The **chain shift** transform [15] is $\mathsf{CS}[f] = f^{\mathsf{CS}}(\text{K}, \mathsf{padCS}(\text{M}))$.

Now we define two new transforms. The **strengthened Chain Shift** transform $\mathsf{sCS}[f] = f^{\mathsf{CS}}(\text{K}, \mathsf{padCS_s}(\text{M}))$ adds strengthening to the CS transform. The enveloped Shoup iteration f^{ESh}: $(\{0,1\}^k \times \{0,1\}^{tn}) \times D^{\circ} \to \{0,1\}^n$ uses an internal f^{Sh} iteration and then an envelope like the one used in f^{CS}. Note that the output of f^{Sh} is xor'd with a key mask and $IV2$ is xor'd with K_0 (which serves to preserve that $IV1 \neq IV2$ across the masking). Then the **Enveloped Shoup** transform is $\mathsf{ESh}[f] = f^{\mathsf{ESh}}(\text{K}, \mathsf{padCS_s}(\text{M}))$.

For a fixed compression function f each transform defines a hash function, e.g. $\mathsf{MD}^f = \mathsf{MD}[f]$ is the hash function with signature MD^f: $\{0,1\}^k \times \{0,1\}^* \to \{0,1\}^n$ defined by $\mathsf{MD}^f(K, M) = f^{+}(K, \mathsf{pad}(M))$.

For each padding function g (and therefore each transform) we define an *efficiency function* τ_g: $\mathbb{N} \to \mathbb{N}$ defined as $\tau_g(L) = \lceil |g(M)|/d \rceil$ for any $M \in \{0,1\}^L$. For brevity we will often just write $\tau(L)$ where the padding function of interest is clear from context. Note that efficiency functions are called application functions in [15]. The efficiency functions are given in Figure 1, where for padPF we utilize the concrete example outlined above.

5 Security Analysis of the Transforms

We now analyze the security of the nine transforms in terms of the five different security goals. The summary of our analysis is given in Figure 1. Some of the results are already established by prior work; we omit discussion of these and refer the reader to the given citations. We discuss each security property in turn. Due to a lack of space, we cannot include any proofs here. A complete treatment, including proofs, is given in the full version of the paper [5].

5.1 Collision Resistance Preservation

Collision-resistance preservation is typically achieved via strengthening: appending the length of a message to it before processing. Not surprisingly, transforms that omit strengthening are *not* CR-Pr: we show this for Pre, NI, and CS. On the other hand, those that include strengthening are CR-Pr, as we show for sNI, sCS, and ESh.

Theorem 1. [**Negative results:** Pre, NI, CS] *Let* f: $\{0,1\}^k \times \{0,1\}^{n+d} \to \{0,1\}^{n-1}$ *be a* $(t, n+d, \epsilon)$*-cr function. Then there exists a function* g: $\{0,1\}^k \times \{0,1\}^{n+d} \to \{0,1\}^n$ *that is* $(t - \mathsf{c}_1, n+d, \epsilon)$*-cr but* $\mathsf{Pre}[g], \mathsf{NI}[g], \mathsf{CS}[g]$ *are at most* $(\mathsf{c}_2, 3d, 1)$*-cr where* $\mathsf{c}_1, \mathsf{c}_2$ *are small constants.* □

Theorem 2. [**Positive results:** sNI, sCS, ESh] *Let* f: $\{0,1\}^k \times \{0,1\}^{n+d} \to \{0,1\}^n$ *be a* $(t, n+d, \epsilon)$*-cr function. Then* $\mathsf{T}[f]$ *is* (t', L, ϵ')*-cr where for*
(1) $\mathsf{T} = \mathsf{sNI}$ *we have* $t' = t - \mathsf{c}_1\tau(L)$ *and* $\epsilon' = \epsilon/2$
(2) $\mathsf{T} \in \{\mathsf{sCS}, \mathsf{ESh}\}$ *we have* $t' = t - \mathsf{c}_2\tau(L)$ *and* $\epsilon' = \epsilon$
where $\mathsf{c}_1, \mathsf{c}_2$ *are small constants.* □

5.2 MAC (Unforgeability) Preservation

We show that MD, sMD, and Sh *do not* preserve unforgeability. Recall that in this setting, the key material (including the key masks of Sh) is secret and therefore unknown to the adversary. While it may not be surprising that MD and sMD are not MAC-Pr, one might be tempted to think that the large amount of secret key material used in Sh could assist in preserving unforgeability. Unfortunately this is not the case, though the counter-example is involved [5]. On the positive side, we have that ESh is MAC-Pr, which can be shown using the proof techniques in [15].

Theorem 3. [**Negative results:** MD, sMD, Sh] *If there exists a function* f: $\{0,1\}^k \times \{0,1\}^{n+d} \to \{0,1\}^{n-1}$ *that is a* $(t, q, n+d, \epsilon)$*-mac, then there exists a function* g: $\{0,1\}^k \times \{0,1\}^{n+d} \to \{0,1\}^n$ *that is a* $(t - \mathsf{c}_1 q, q, n+d, \epsilon)$*-mac but*
(1) $\mathsf{MD}[g]$, $\mathsf{sMD}[f']$ *are at most* $(\mathsf{c}_2, n-1, 3d, 1/2)$*-mac*
(2) $\mathsf{Sh}[g]$ *is at most a* $(\mathsf{c}_3, 2(n-1), 3d, 1/4)$*-mac*
where c_1, c_2, *and* c_3 *are small constants.* □

Theorem 4. [**Positive results:** ESh] *Let* f: $\{0,1\}^k \times \{0,1\}^{n+d} \to \{0,1\}^n$ *be a* (t, q, ϵ)*-mac. Then* $\mathsf{ESh}[f]$ *is a* $(t - \mathsf{c}(q'+1)\tau(L), q', L, \epsilon')$*-mac where*

$$q' = (q - \tau(L) + 1)/\tau(L) \quad \text{and} \quad \epsilon' = (q^2/2 + 3q/2 + 1)\epsilon$$

for c *a small constant and any* $\{K_i\}_1^t \in \{0,1\}^{tn}$ *with* $t = \log_2(\tau(L))$. □

5.3 Pseudorandom Function Preservation

In the non-dedicated-key setting, building PRF preserving transforms is non-trivial and the proofs of security are complex. In stark contrast to this, we show that all of the dedicated-key transforms considered here are PRF-Pr, and the proofs establishing this are relatively straightforward (see [5]).

Theorem 5. [**Positive results:** MD, sMD, Pre, Sh, sNI, NI, CS, sCS, ESh] *Let* f: $\{0,1\}^k \times \{0,1\}^{n+d} \to \{0,1\}^n$ *be a* $(t, q, n+d, \epsilon)$*-prf. Then* $\mathsf{T}[f]$ *is a* (t', q', L, ϵ')*-prf where for*
(1) $\mathsf{T} \in \{\mathsf{MD}, \mathsf{sMD}, \mathsf{Pre}, \mathsf{Sh}, \mathsf{CS}, \mathsf{sCS}, \mathsf{ESh}\}$, $t' = t - \mathsf{c}q\tau(L)$, $q' = q/\tau(L)$, $\epsilon' = \epsilon + q^2\tau(L)^2/2^n$,
(2) $\mathsf{T} \in \{\mathsf{sNI}, \mathsf{NI}\}$, $t' = t - \mathsf{c}q\tau(L)$, $q' = q/\tau(L)$, $\epsilon' = 2\epsilon + q^2(\tau(L)-1)^2/2^n$
where c *is a small constant and* $\{K_i\}_1^t \in \{0,1\}^{tn}$ *for* $t = \log_2(\tau(L))$. □

5.4 Pseudorandom Oracle Preservation

Establishing that a transform is PRO-Pr ensures that a hash function constructed with it "behaves like a random oracle" under the assumption that the compression function is ideal. This is important for usage of hash functions to instantiate random oracles, as discussed at length in [11]. To reason about dedicated-key transforms, we model a compression function $f\colon \{0,1\}^k \times \{0,1\}^{n+d} \to \{0,1\}^n$ as a family of random oracles, one for each key in $\{0,1\}^k$. However, since we only ever use one or two fixed keys from $\{0,1\}^k$, we will (without loss of generality) simply utilize two separate random oracles $f = \mathsf{RF}_{n+d,n}$ and $g = \mathsf{RF}_{n+d,n}$ from the family. This simplifies our analysis, and, in particular, means that many results from the keyless setting carry over directly to the dedicated-key setting (see Figure 1). For example, the negative results that MD^f and sMD^f are not PROs follows from simple length-extension attacks (see [11]). The security of CS, and sCS is implied by the security of EMD [6] and the security of sNI and NI is implied by results in [11].

We point out that Sh^f is not a PRO. Since the key masks are public, simple length extension attacks enable an adversary to differentiate between it and a true variable-input-length random oracle. On the other hand ESh^f is a PRO.

Theorem 6. [**Negative result:** Sh] *Let $f = \mathsf{RF}_{n+d,n}$ be a random oracle. Then there exists an $(\mathsf{c}, t_{\mathcal{S}}, 2, 1, 2d, 1-2^{-n})$-pro adversary $\mathcal{A}$ against Sh^f for any simulator $\mathcal{S}$ with arbitrary running time $t_{\mathcal{S}}$. The running time of $\mathcal{A}$ is a small constant* c. □

Theorem 7. [**Positive result:** ESh] *Let $f = \mathsf{RF}_{n+d,n}$ be a random oracle. Then ESh^f is a $(t_{\mathcal{A}}, t_{\mathcal{S}}, q_1, (q_2+q_3), L, \epsilon)$-pro where*

$$\epsilon \leq (l^2 q_1^2 + (lq_1 + q_2)(q_2 + q_3))/2^n + lq_1/2^n$$

for $l = \tau(L)$. The running time $t_{\mathcal{A}}$ is arbitrary while $t_{\mathcal{S}} = \mathcal{O}(q_2^2 + q_2 q_3)$. □

5.5 Target Collision Resistance Preservation

Universal one-way hash functions (UOWHF) were first introduced by Naor and Reingold [18] and later went by the term target collision resistance (TCR) [10]. The best TCR-Pr transforms known require a logarithmic (in the size of the messages allowed) amount of key material. Mironov has given strong evidence that one cannot do better [17]. Due to this and because any CR function is also TCR [23], it might be sufficient in most settings to utilize a dedicated-key transform that preserves the four previous properties. Still, target-collision resistant functions are useful in some settings [10] and establishing their security based only on the compression function being TCR results in a stronger guarantee of security (this is analogous to the discussion of MAC-Pr versus PRF-Pr in Section 3). Thus, we extend our analysis to this property.

In light of Mironov's result, it is not surprising that Pre, sNI, NI, CS, and sCS are not TCR-Pr, though we establish these facts directly. On the other hand, a

proof that ESh is TCR-Pr can be straightforwardly derived from the proof that Sh is TCR (see [17] or [24]).

Theorem 8. **[Negative results: Pre, sNI, NI, CS, sCS]** *If there exists a function* $f\colon \{0,1\}^k \times \{0,1\}^{n+d} \to \{0,1\}^{n-1}$ *that is* $(t, n+d, \epsilon)$*-tcr, then there exists a function* $g\colon \{0,1\}^k \times \{0,1\}^{(n+d} \to \{0,1\}^n$ *that is* $(t - \mathsf{c}_1, n+d, \epsilon + 2^{-k+1})$*-tcr but* $\mathsf{Pre}[g], \mathsf{sNI}[g], \mathsf{NI}[g], \mathsf{CS}[g], \mathsf{sCS}[g]$ *are at most* $(\mathsf{c}_2, 3d, 1-2^{-k})$*-tcr where* $\mathsf{c}_1, \mathsf{c}_2$ *are small constants.* □

Theorem 9. **[Positive result: ESh]** *Let* $f\colon \{0,1\}^k \times \{0,1\}^{n+d} \to \{0,1\}^n$ *be* $(t, n+d, \epsilon)$*-tcr. Then* $\mathsf{ESh}[f]$ *is* $(t - \mathsf{c}t_f^2\tau(L)^2, L, \epsilon\tau(L))$*-tcr for a small constant* c *and where* T_f *is the time to execute* f*.* □

Acknowledgments

The authors are supported in part by NSF grants CNS 0524765, CNS 0627779, and a gift from Intel Corporation.

References

1. An, J., Bellare, M.: Constructing VIL-MACs from FIL-MACs: Message authentication under weakened assumptions. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 252–269. Springer, Heidelberg (1999)
2. Bellare, M.: New Proofs for NMAC and HMAC: Security Without Collision-Resistance. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 113–120. Springer, Heidelberg (2006)
3. Bellare, M., Canetti, R., Krawczyk, H.: Keying Hash Functions for Message Authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996)
4. Bellare, M., Canetti, R., Krawczyk, H.: Pseudorandom functions revisited: the cascade construction and its concrete security. In: Proceedings of the 37th Annual Symposium on Foundations of Computer Science – FOCS '96, IEEE Computer Society, pp. 514–523. IEEE Computer Society Press, Los Alamitos (1996)
5. Bellare, M., Ristenpart, T.: Hash Functions in the Dedicated-Key Setting: Design Choices and MPP Transforms (2007), Full version of current paper, `http://www.cse.ucsd.edu/users/mihir/`
6. Bellare, M., Ristenpart, T.: Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 299–314. Springer, Heidelberg (2006)
7. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. First ACM Conference on Computer and Communications Security – CCS '93, pp. 62–73. ACM Press, New York (1993)
8. Bellare, M., Rogaway, P.: Optimal Asymmetric Encryption. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 92–111. Springer, Heidelberg (1995)
9. Bellare, M., Rogaway, P.: The Exact Security of Digital Signatures - How to Sign with RSA and Rabin. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (1996)

10. Bellare, M., Rogaway, P.: Collision-Resistant Hashing: Towards Making UOWHFs Practical. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 470–484. Springer, Heidelberg (1997)
11. Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård Revisited: How to Construct a Hash Function. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 21–39. Springer, Heidelberg (2005)
12. Damgård, I.: A design principle for hash functions. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer, Heidelberg (1990)
13. Maurer, U., Renner, R., Holenstein, C.: Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 21–39. Springer, Heidelberg (2004)
14. Maurer, U., Sjödin, J.: Domain Expansion of MACs: Alternative Uses of the FIL-MAC. In: Smart, N.P. (ed.) Cryptography and Coding. LNCS, vol. 3796, pp. 168–185. Springer, Heidelberg (2005)
15. Maurer, U., Sjödin, J.: Single-key AIL-MACs from any FIL-MAC. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 472–484. Springer, Heidelberg (2005)
16. Merkle, R.: One way hash functions and DES. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 428–446. Springer, Heidelberg (1990)
17. Mironov, I.: Hash functions: from Merkle-Damgård to Shoup. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 166–181. Springer, Heidelberg (2001)
18. Naor, M., Yung, M.: Universal one-way hash functions and their cryptographic applications. In: Proceedings of the 21st Annual ACM Symposium on Theory of Computing – STOC'89, pp. 33–43. ACM Press, New York (1989)
19. National Institute of Standards and Technology. FIPS PUB 180-1: Secure Hash Standard. Supersedes FIPS PUB 180 1993 May 11 (1995)
20. RSA Laboratories. RSA PKCS #1 v2.1: RSA Cryptography Standards (2002)
21. Rivest, R.: The MD4 Message Digest Algorithm. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 303–311. Springer, Heidelberg (1991)
22. Rogaway, P.: Formalizing Human Ignorance: Collision-Resistant Hashing without the Keys. In: Nguyen, P.Q. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 221–228. Springer, Heidelberg (2006)
23. Rogaway, P., Shrimpton, T.: Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 371–388. Springer, Heidelberg (2004)
24. Shoup, V.: A Composition Theorem for Universal One-Way Hash Functions. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 445–452. Springer, Heidelberg (2000)
25. Tsudik, G.: Message Authentication with One-way Hash Functions. SIGCOMM Comp. Commun. Rev. 22(5), 29–38 (1992)
26. Wang, X., Yin, Y., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
27. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)