

A preliminary version of this paper appears in *International Colloquium on Automata, Languages, and Programming – ICALP 07*, Lecture Notes in Computer Science Vol. 4596, pp. 399–410, L. Arge et al. ed., Springer-Verlag, 2007. This is the full version.

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

MIHIR BELLARE \*    THOMAS RISTENPART†

October 2007

## Abstract

In the dedicated-key setting, one starts with a compression function  $f: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$  and builds a family of hash functions  $H^f: \mathcal{K} \times \mathcal{M} \rightarrow \{0, 1\}^n$  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.

---

\*Dept. of Computer Science & Engineering 0404, University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0404, USA. Email: [mihir@cs.ucsd.edu](mailto:mihir@cs.ucsd.edu). URL: <http://www-cse.ucsd.edu/users/mihir>. Supported in part by NSF grant CNS 0524765 and a gift from Intel Corporation.

†Dept. of Computer Science & Engineering 0404, University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0404, USA. Email: [tristenp@cs.ucsd.edu](mailto:tristenp@cs.ucsd.edu). URL: <http://www-cse.ucsd.edu/users/tristenp>. Supported in part by above-mentioned grants of first author.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Notation and Definitions</b>	<b>4</b>
<b>3</b>	<b>Hash Functions in the Dedicated Key Setting</b>	<b>5</b>
<b>4</b>	<b>Dedicated Key Transforms</b>	<b>7</b>
<b>5</b>	<b>Security Analysis of the Transforms</b>	<b>9</b>
5.1	Collision Resistance Preservation . . . . .	10
5.2	MAC (Unforgeability) Preservation . . . . .	12
5.3	Pseudorandom Function Preservation . . . . .	16
5.4	Pseudorandom Oracle Preservation . . . . .	17
5.5	Target Collision Resistance Preservation . . . . .	20
<b>A</b>	<b>Proof of Lemma 5.12</b>	<b>24</b>

# 1 Introduction

**TWO SETTINGS.** A popular method for designing hash functions proceeds as follows. First, one designs a compression function  $f: \{0,1\}^{d+n} \rightarrow \{0,1\}^n$ , 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  $H^f: \mathcal{M} \rightarrow \{0,1\}^n$  associated to  $f$ , where  $\mathcal{M}$  is some large message space. Most in-use hash functions, for example the MD-x [24] family and SHA-1 [22], 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  $f: \{0,1\}^k \times \{0,1\}^{d+n} \rightarrow \{0,1\}^n$ . A transform  $H$  now uses  $f(\cdot, \cdot)$  as a black-box to implement a family of hash functions  $H^f: \mathcal{K} \times \mathcal{M} \rightarrow \{0,1\}^n$  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 publically 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 [30, 31], new hash functions are going to be designed 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. We present a discussion of the relative merits of the dedicated-key setting, but also its most significant drawbacks.

If one chooses 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 [5]. 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. We investigate nine transforms, two of which are novel, determining if they successfully preserve each property  $P$  of interest.

We now briefly summarize our results in more detail.

**THE DEDICATED-KEY SETTING.** In Section 3, we present relative merits of the dedicated key setting compared to the more traditional setting. On the positive side, we highlight some practical and concrete benefits of the dedicated key setting in addition to the more widely acknowledged theoretical benefits. The dedicated key setting enables hash function heterogeneity (allowing users to specify independent instances of the hash function) and also improves 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) [19, 12], strengthened MD (sMD) [12], prefix-free MD (Pre) [18], Shoup’s transform (Sh) [28], the strengthened Nested Iteration transform (sNI) [1], the Nested Iteration transform (NI) [18], and the Chain-Shift transform (CS) [18]. 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). Each property is important for a widely-used application of hash functions; see Section 5 for a more detailed discussion. 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

	CR-Pr	MAC-Pr	PRF-Pr	PRO-Pr	TCR-Pr	Efficiency $\tau(L)$	Key bits
MD	No [19, 12]	<b>No</b>	<b>Yes</b>	No [11]	No [9]	$\lceil (L+1)/d \rceil$	$k$
sMD	Yes [19, 12]	<b>No</b>	<b>Yes</b>	No [11]	No [9]	$\lceil (L+65)/d \rceil$	$k$
Pre	<b>No</b>	Yes [18]	<b>Yes</b>	Yes [11]	<b>No</b>	$\lceil (L+1)/(d-1) \rceil$	$k$
Sh	Yes [28]	<b>No</b>	<b>Yes</b>	<b>No</b>	Yes [28]	$\lceil (L+65)/d \rceil$	$k + \zeta(L)$
sNI	<b>Yes</b>	Yes [1]	<b>Yes</b>	Yes [5]	<b>No</b>	$\lceil (L+65)/d \rceil$	$2k$
NI	<b>No</b>	Yes [18]	<b>Yes</b>	Yes [5]	<b>No</b>	$\lceil (L+1)/d \rceil$	$2k$
CS	<b>No</b>	Yes [18]	<b>Yes</b>	Yes [5]	<b>No</b>	$\lceil (L+1+n)/d \rceil$	$k$
sCS	<b>Yes</b>	Yes [18]	<b>Yes</b>	Yes [5]	<b>No</b>	$\lceil (L+65+n)/d \rceil$	$k$
ESh	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>	$\lceil (L+65+n)/d \rceil$	$k + \omega(L)$

Figure 1: Summary of transforms in the dedicated-key setting when applied to a compression function  $f: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$ . Bold-faced claims are novel. Efficiency is measured by  $\tau(L)$ , the number of compression function applications used to hash an  $L$ -bit string. Key bits is the number of bits of key material required to hash an  $L$ -bit string. The functions  $\zeta(L)$  and  $\omega(L)$  are defined as  $n\lceil \log_2 \tau(L) \rceil$  and  $n(\lceil \log_2(\tau(L) - 1) \rceil + 1)$  for the appropriate  $\tau(L)$ .

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 [21], which we'll call target-collision resistance (following [9]). Preserving this property is potentially of less practical interest, due to the need for more key material; see the discussion in Section 5. 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.

## 2 Notation and Definitions

NOTATION. We denote pairwise concatenation by  $\parallel$ , e.g.  $M \parallel M'$ , and write  $M_1 \cdots M_k$  to mean  $M_1 \parallel M_2 \parallel \cdots \parallel M_k$ . The  $i^{\text{th}}$  bit of a string  $M$  is  $M[i]$  and so  $M = M[1] \parallel \cdots \parallel M[|M|]$ . For brevity, we define the following semantics for the notation  $M_1 \cdots M_k \stackrel{d}{\dashv} M$  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  $M_1, M_2, \dots, M_k$  where  $|M_i| = d$  for  $1 \leq i \leq k$ , otherwise parse  $M$  into  $M_1, M_2, \dots, M_{k-1}, M_k$  where  $|M_i| = d$  for  $1 \leq i \leq k-1$  and  $|M_k| = |M| \bmod d$ . For any finite set  $S$  we write  $s \stackrel{\$}{\leftarrow} S$  to signify uniformly choosing a value  $s \in S$ . We write  $s \stackrel{\$}{\leftarrow} \mathcal{A}(x_1, x_2, \dots)$  to mean assign to  $s$  the result of running  $\mathcal{A}$  with fresh random coins on inputs  $x_1, x_2, \dots$ . A random oracle is an algorithm  $\text{RF}_{\text{Dom}, \text{Rng}}$  that, on input  $X \in \text{Dom}$ , returns a value  $Y \stackrel{\$}{\leftarrow} \text{Rng}$ . Repeat queries are, however, answered consistently. We sometimes write  $\text{RF}_{d,r}$  when  $\text{Dom} = \{0, 1\}^d$  and  $\text{Rng} = \{0, 1\}^r$ .

SECURITY NOTIONS. We recall the security definitions needed in the rest of the paper. Let  $F: \mathcal{K} \times \text{Dom} \rightarrow \text{Rng}$  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:  $\epsilon = \Pr \left[ (X, S) \stackrel{\$}{\leftarrow} \mathcal{A}_1, K \stackrel{\$}{\leftarrow} \mathcal{K}, X' \stackrel{\$}{\leftarrow} \mathcal{A}_2(S, K) : \begin{array}{l} X \neq X' \wedge \\ F_K(X) = F_K(X') \end{array} \right]$
- cr:  $\epsilon = \Pr \left[ K \stackrel{\$}{\leftarrow} \mathcal{K}, (X, X') \stackrel{\$}{\leftarrow} \mathcal{A}(K) : X \neq X' \wedge F_K(X) = F_K(X') \right]$

- 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:  $\epsilon = \Pr \left[ K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{F_K(\cdot)} \Rightarrow 1 \right] - \Pr \left[ \rho \xleftarrow{\$} \text{Func}(Dom, Rng) : \mathcal{A}^{\rho(\cdot)} \Rightarrow 1 \right]$

where the probabilities are over the specified random choices and the coins used by  $\mathcal{A}$ . The set  $\text{Func}(Dom, Rng)$  includes all functions  $\rho: Dom \rightarrow Rng$ . In the tcr game  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$  is a pair of algorithms. Now letting  $F$  be an algorithm given oracle access to an ideal compression function  $f = \text{RF}_{n+d,n}$  we define the last security experiment:

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

where the probabilities are over the specified random choices, the coins used by  $\mathcal{A}$  and  $\mathcal{S}$ , and the coins used by  $\mathcal{F} = \text{RF}_{Dom, Rng}$  and  $f = \text{RF}_{n+d,n}$ . The simulator  $\mathcal{S}$  maintains state across queries and has oracle access to  $\mathcal{F}$ . For more details on the pseudorandom oracle definition see [5, 11, 16].

We say that  $F$  is  $(t, L, \epsilon)$ -xxx for  $\text{xxx} \in \{\text{tcr}, \text{cr}\}$  if any adversary  $\mathcal{A}$  running in time at most  $t$  and outputting messages of length less than or equal to  $L$  bits has  $\epsilon$  probability of success in the xxx game. Similarly we say that  $F$  is a  $(t, q, L, \epsilon)$ -xxx for  $\text{xxx} \in \{\text{mac}, \text{prf}\}$  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  $\epsilon$  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 for 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, there exists a simulator  $\mathcal{S}$  running in time  $t_{\mathcal{S}}$  such that  $\mathcal{A}$ 's probability of success in the pro game is at most  $\epsilon$ .

### 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*, in which we can utilize numerous different hash function instances. To understand why this is useful for security, we discuss (as an example) an important application of publically-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  $H^h = \text{SHA-1}$  [30], 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  $H^h: \mathcal{K} \times \mathcal{M} \rightarrow \{0, 1\}^n$  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 [30], 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  $H^h: \mathcal{M} \rightarrow \{0, 1\}^n$  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 publically.

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 (e.g., prefix-free MD [4], EMD [5], and NMAC [3, 2], etc.). 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  $h: \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$  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, 18, 17]. This allows us to utilize hash functions as MACs under just the assumption that  $h: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$  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.

**KEYING AND COLLISION-RESISTANCE.** Hash functions with dedicated key inputs are an easy solution for the *foundations-of-hashing dilemma* [25], which is a problem of theoretical interest. The dilemma refers to the fact that  $h: \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$  can *not* be collision-resistant: by the pigeonhole principle there are two strings  $X, X'$  both of length  $n + 1$  bits 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 [25], 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  $h: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$ , 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  $H: \{0, 1\}^* \rightarrow \{0, 1\}^n$ . We point out that it is easy to allow such compatibility when starting with a dedicated-key hash function  $H': \mathcal{K} \times \{0, 1\}^* \rightarrow \{0, 1\}^n$ . Simply fix an honestly generated and publically-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.** A dedicated-key hash function's security guarantees only hold in situations where the key is generated honestly. This is typically not a concern because 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 keys. Still, given the wide-spread use of hash functions in non-standard settings, one should be aware of the potential for abuse in applications that require security even in the face of adversarially-chosen keys. A simple solution for such settings would be to require a fixed, honestly-generated key as mentioned above.

## 4 Dedicated Key Transforms

Let  $f: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$  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  $H^f: \mathcal{K} \times \mathcal{M} \rightarrow \{0, 1\}^n$ . 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  $D^\circ$ . We define the following padding functions:

- **pad**:  $\{0, 1\}^* \rightarrow D^+$  is defined by  $\text{pad}(M) = M \parallel 10^r$
- **pad<sub>s</sub>**:  $\mathcal{D} \rightarrow D^+$  is defined by  $\text{pad}_s(M) = M \parallel 10^r \parallel \langle |M| \rangle_{64}$
- **padPF**:  $\{0, 1\}^* \rightarrow D^+$  is a prefix-free padding function: for any  $M, M' \in \{0, 1\}^*$  where  $|M| < |M'|$  we have that  $\text{padPF}(M)$  is not a prefix of  $\text{padPF}(M')$ . For the rest of the paper we fix the following concrete realization of a prefix-free encoding. Pad the message with  $10^r$  for minimal value  $r$  such that the resulting string has length a multiple of  $d - 1$  bits. Parse the resulting string into blocks of  $d - 1$  bits, add a zero to each block except the final block, which has a one added to it, and output the result.
- **padCS**:  $\{0, 1\}^* \rightarrow D^\circ$  is defined by  $\text{padCS}(M) = M \parallel 10^r$
- **padCS<sub>s</sub>**:  $\mathcal{D} \rightarrow D^\circ$  is defined by  $\text{padCS}_s(M) = M \parallel 10^r \parallel \langle |M| \rangle_{64} \parallel 0^p$

where for **pad**, **pad<sub>s</sub>**, 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 **padCS<sub>s</sub>** we define  $r$  and  $p$  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 needed to make the returned string in  $D^\circ$ . 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 and two new transforms.

**Plain, Strengthened, and Prefix-free MD.** The Merkle-Damgård (MD) iteration  $f^+: \{0, 1\}^k \times D^+ \rightarrow \{0, 1\}^n$  repeatedly applies  $f$ . We define the following transforms using the MD iteration.

$$\begin{array}{ll} \text{Plain MD [19, 12]:} & \text{MD}[f] = f^+(\kappa, \text{pad}(M)) \\ \text{Strengthened MD [12]:} & \text{sMD}[f] = f^+(\kappa, \text{pad}_s(M)) \\ \text{Prefix-free MD [18]:} & \text{Pre}[f] = f^+(\kappa, \text{padPF}(M)) \end{array}$$

The placeholders  $\kappa$  and  $M$  designate how to handle the key and message inputs.

**Shoup.** The Shoup iteration  $f^{\text{Sh}}: (\{0, 1\}^k \times \{0, 1\}^{\kappa n}) \times D^+ \rightarrow \{0, 1\}^n$  utilizes  $\kappa = \lceil \log_2(\sigma) \rceil$  key masks where  $\sigma$  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  $\{K_i\}_1^\kappa \in \{0, 1\}^{\kappa n}$  are  $\kappa$   $n$ -bit keys that are used to ‘mask’ chaining variable values with secret key material. We define the Shoup transform as follows.

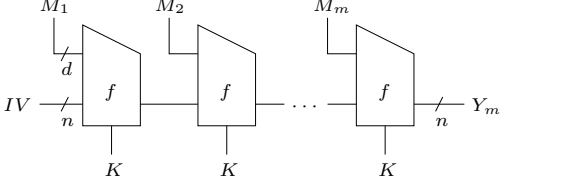
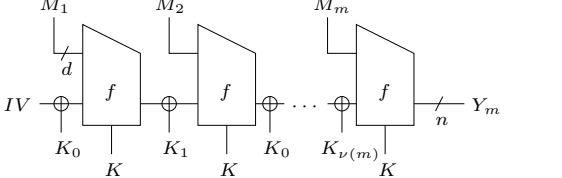
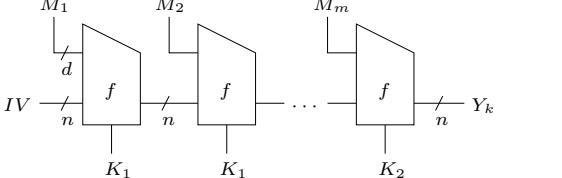
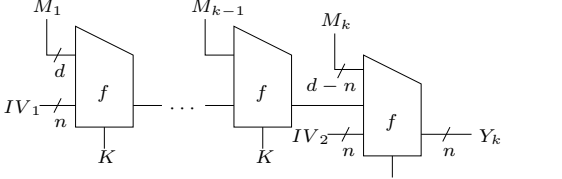
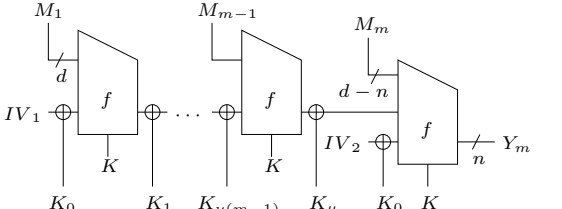
<p><b>Algorithm <math>f^+(K, M)</math>:</b>  <math>M_1 \cdots M_m \stackrel{d}{\leftarrow} M</math>; <math>Y_0 \leftarrow IV</math>  <b>for</b> <math>i = 1</math> <b>to</b> <math>m</math> <b>do</b>  <math>Y_i \leftarrow f_K(Y_{i-1} \parallel M_i)</math>  <b>Ret</b> <math>Y_m</math></p>	
<p><b>Algorithm <math>f^{\text{Sh}}((K, \{K_i\}_1^\kappa), M)</math>:</b>  <math>M_1 \cdots M_m \stackrel{d}{\leftarrow} M</math>; <math>Y_0 \leftarrow IV</math>  <b>for</b> <math>i = 1</math> <b>to</b> <math>m</math> <b>do</b>  <math>Y_i \leftarrow f_K(Y_{i-1} \oplus K_{\nu(i)} \parallel M_i)</math>  <b>Ret</b> <math>Y_m</math></p>	
<p><b>Algorithm <math>f^{\text{NI}}((K_1, K_2), M)</math>:</b>  <math>M_1 \cdots M_m \stackrel{d}{\leftarrow} M</math>  <math>Y_{m-1} \leftarrow f^+(K_1, M_1 \cdots M_{m-1})</math>  <b>Ret</b> <math>Y_m \leftarrow f_{K_2}(Y_{m-1} \parallel M_m)</math></p>	
<p><b>Algorithm <math>f^{\text{CS}}(K, M)</math>:</b>  <math>M_1 \cdots M_m \stackrel{d}{\leftarrow} M</math>  <math>Y_{m-1} \leftarrow f^+(K, M_1 \cdots M_{m-1})</math>  <b>Ret</b> <math>f_K(IV_2 \parallel Y_{m-1} \parallel M_m)</math></p>	
<p><b>Algorithm <math>f^{\text{Esh}}((K, \{K_i\}_1^\mu), M)</math>:</b>  <math>M_1 \cdots M_m \stackrel{d}{\leftarrow} M</math>; <math>Y_0 \leftarrow IV_1</math>  <math>Y_{m-1} \leftarrow f^{\text{Sh}}((K, \{K_i\}_1^{t-1}), M_1 \cdots M_{m-1})</math>  <b>Ret</b> <math>f_K((IV_2 \oplus K_0) \parallel (Y_{m-1} \oplus K_\mu) \parallel M_m)</math></p>	

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

$$\text{Shoup [28]: } \text{Sh}[f] = f^{\text{Sh}}(\kappa, \text{pad}_s(M))$$

**(Strengthened) Nested Iteration.** The nested iteration  $f^{\text{NI}}: (\{0, 1\}^k \times \{0, 1\}^k) \times D^+ \rightarrow \{0, 1\}^n$  just envelopes an  $f^+$  iteration with an application of  $f$  using a second key. We define the following two transforms.

$$\begin{aligned} \text{Strengthened Nested Iteration [1]: } \quad \text{sNI}[f] &= f^{\text{NI}}(\kappa, \text{pad}_s(M)) \\ \text{Nested Iteration [18]:} \quad \text{NI}[f] &= f^{\text{NI}}(\kappa, \text{pad}(M)) \end{aligned}$$

**Chain Shift.** The chain shift iteration  $f^{\text{CS}}: \{0, 1\}^k \times D^\circ \rightarrow \{0, 1\}^n$ , envelopes an internal  $f^+$  iteration with an application of  $f(IV_2 \parallel \cdot)$ . We require that  $IV_2 \neq IV_1$ . Then we have the following transform.

$$\text{Chain Shift [18]: } \text{CS}[f] = f^{\text{CS}}(\kappa, \text{padCS}(M))$$

**New transforms.** Now we define two new transforms.



$$\begin{aligned} \text{Strengthened Chain Shift:} & \quad \text{CS}[f] = f^{\text{CS}}(K, \text{padCS}_s(M)) \\ \text{Enveloped Shoup:} & \quad \text{ESh}[f] = f^{\text{ESh}}(K, \text{padCS}_s(M)) \end{aligned}$$

The strengthened Chain Shift transform adds strengthening to the CS transform. The Enveloped Shoup transform applies an internal Shoup iteration and then envelopes the result. It requires  $\mu = \lceil \log_2 \sigma \rceil + 1$   $n$ -bit key masks where  $\sigma$  is the maximum number of internal iterations of  $f$  allowed. Note that the key mask schedule has  $K_0$  used for both  $IV_1$  and  $IV_2$ . This serves to preserve the uniqueness of the two initialization vectors across the xor operations, which we leverage in the proof that ESh is PRO-Pr. Additionally, the key mask  $K_\mu$  is only used for the chaining variable fed into the envelope, which means that  $K_\mu - 1$  masks are available for the internal Shoup iteration. The distinct key mask for the envelope is important for ensuring (target) collision resistance preservation in the case that  $d < n + 64$  (in this case the strengthening does not fit in the envelope and so must be placed in the second to last compression function application).

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

For each padding function  $g$  (and therefore each transform) we define an *efficiency function*  $\tau_g: \mathbb{N} \rightarrow \mathbb{N}$  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 [18]. Figure 1 lists the the efficiency functions of the nine transforms.

## 5 Security Analysis of the Transforms

In this section we give a concrete security treatment of the nine transforms in terms of the five different security goals identified in the introduction. We investigate each transform for each of the five properties. For each transform, property pair we seek either a counter-example (a compression function for which the transform constructs a hash function without the property in question) or a proof that the property is preserved by the transform. Some results are already established by prior work, see Figure 1 for citations. The next few sections contain detailed analyses to establish the new results. First, we discuss each of the properties and motivate their importance and then give a brief summary of the results and their implications.

**MPP TRANSFORMS.** As observed in [5], current hash function usage spans a wide range of applications, many with disjoint security requirements. For example, hash functions are currently simultaneously utilized for being CR and for instantiating random oracles (in schemes such as RSA-OAEP [7] and RSA-PSS [8] specified in the RSA PKCS#1 v2.1 standard [23]). Hash functions are also keyed and used as message authentication codes (MACs) and pseudorandom functions (PRFs). (For example, HMAC [3], a popular hash-function based construction, is used for message authentication in SSH, IPsec [15], and TLS [13] and for key derivation, where it must be a PRF, in TLS [13] and IKE [14].) Hash function design should reflect such usage, and for transforms this means being *multi-property-preserving*: a transform should simultaneously preserve many properties of interest. This enables building a *single* hash function that can be used for a variety of applications, easing the burden of implementation and standardization. To be useful for all the applications listed above, we therefore ask that our transforms be collision-resistance preserving (CR-Pr), MAC preserving (MAC-Pr), PRF preserving (PRF-Pr), and pseudorandom-oracle preserving (PRO-Pr).

Besides these four properties, one can also ask for a fifth, namely target collision resistance (TCR). This has applications in the setting of some digital signature schemes, for a discussion see [9]. Unfortunately, the best transforms (Sh being one) require a significant number of key bits, in fact logarithmic in the number of blocks hashed. To make matters worse, there is strong evidence that we're not likely to do much better with MD-style transforms [20, 27]. Nevertheless we add this property to the list, and investigate which transforms are target collision-resistance preserving (TCR-Pr) in Section 5.5.

**SUMMARY OF RESULTS AND DISCUSSION.** The summary of our analysis is given in Figure 1. It shows that only the strengthened Nested Iteration (sNI) transform simultaneously preserves CR, MAC, PRF, and PRO. However it requires two keys, and one can do better with the (new) strengthened Chain Shift (sCS) transform. This last just adds strengthening to CS. If TCR is added to the list, then none of the transforms, including sCS, preserve all five properties. Hence the Enveloped Shoup transform which accomplishes just that. Note that the two new transforms only differ in the masks; in particular ESh with all the masks set to zero bits is exactly sCS.

## 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 5.1 [Negative results: Pre, NI, CS]** *If there exists a function  $f: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^{n-1}$  that is  $(t, n + d, \epsilon)$ -cr, then there exists a function  $g: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$  that is  $(t - c_1, n + d, \epsilon)$ -cr but  $\text{Pre}[g], \text{NI}[g], \text{CS}[g]$  are at most  $(c_2, 3d - 3, 1)$ -cr where  $c_1, c_2$  are small constants.  $\square$*

**Proof:** We lift the counter-example from [5] to the dedicated-key setting. Without loss of generality let  $IV = 0^n$ . Let  $f: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^{n-1}$  be  $(t, n + d, \epsilon)$ -cr. Then we construct a function  $g: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$  defined by

$$g(K, M) = \begin{cases} 0^n & \text{if } M = 0^n \parallel 0^d \\ f(K, M) \parallel 1 & \text{otherwise} \end{cases}.$$

Then we can see that  $g$  is  $(t - c_1, n + d, \epsilon)$ -cr via a standard argument. Now we give adversaries attacking the hash functions constructed via  $\text{Pre}^g$ ,  $\text{NI}^g$ , and  $\text{CS}^g$ . For  $\text{Pre}^g$  simply output  $0^{d-1} \parallel 0^{d-1}$  and  $0^{d-1} \parallel 0^{d-1} \parallel 0^{d-1}$ . For  $\text{NI}^g$  and  $\text{CS}^g$  have the adversary output  $0^d$  and  $0^d \parallel 0^d$ . These pairs of messages directly lead to collisions under the respective hash functions built using  $g$ .  $\blacksquare$

**Theorem 5.2 [Positive results: sNI, sCS, ESh]** *Let  $f: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$  be a  $(t, n + d, \epsilon)$ -cr function. Then  $T[f]$  is  $(t', L, \epsilon')$ -cr where for*

- (1)  $T = \text{sNI}$  we have  $t' = t - c_1 T_f \tau(L)$  and  $\epsilon' = 2\epsilon$
- (2)  $T \in \{\text{sCS}, \text{ESh}\}$  we have  $t' = t - c_2 T_f \tau(L)$  and  $\epsilon' = \epsilon$

where  $c_1, c_2$  are small constants and  $T_f$  is the time to compute  $f$ .  $\square$

**Proof:** For part (1), let  $\mathcal{A}$  be an  $(t', L, \epsilon')$ -cr adversary against  $\text{sNI}^f$ . Let  $\mathcal{B}_1$  and  $\mathcal{B}_2$  be the two cr adversaries against  $f$  defined below.

Adversary  $\mathcal{B}_1(K)$

$K_2 \xleftarrow{\$} \{0, 1\}^k; (M, M') \xleftarrow{\$} \mathcal{A}(K, K_2)$   
 $P_1 \dots P_m \xleftarrow{d} \text{pad}_s(M); P'_1 \dots P'_{m'} \xleftarrow{d} \text{pad}_s(M')$   
 $Y_0 \leftarrow Y'_0 \leftarrow IV$   
For  $i \in [1..m-1]$  do  $Y_i \leftarrow f_K(Y_{i-1} \parallel P_i)$   
For  $i \in [1..m'-1]$  do  $Y'_i \leftarrow f_K(Y'_{i-1} \parallel P'_i)$   
Let  $\delta$  be smallest value s.t.  
 $Y_{m-\delta-1} \parallel P_{m-\delta} \neq Y'_{m'-\delta-1} \parallel P'_{m'-\delta}$   
If  $\delta = 0$  then Ret  $\perp$   
Ret  $(Y_{m-\delta-1} \parallel P_{m-\delta}, Y'_{m'-\delta-1} \parallel P'_{m'-\delta})$

Adversary  $\mathcal{B}_2(K)$

$K_1 \xleftarrow{\$} \{0, 1\}^k; (M, M') \xleftarrow{\$} \mathcal{A}(K_1, K)$   
 $P_1 \dots P_m \xleftarrow{d} \text{pad}_s(M); P'_1 \dots P'_{m'} \xleftarrow{d} \text{pad}_s(M')$   
 $Y_0 \leftarrow Y'_0 \leftarrow IV$   
For  $i \in [1..m-1]$  do  $Y_i \leftarrow f_K(Y_{i-1} \parallel P_i)$   
For  $i \in [1..m'-1]$  do  $Y'_i \leftarrow f_K(Y'_{i-1} \parallel P'_i)$   
Let  $\delta$  be smallest value s.t.  
 $Y_{m-\delta-1} \parallel P_{m-\delta} \neq Y'_{m'-\delta-1} \parallel P'_{m'-\delta}$   
If  $\delta \neq 0$  then Ret  $\perp$   
Ret  $(Y_{m-1} \parallel P_m, Y'_{m'-1} \parallel P'_{m'})$

Adversary  $\mathcal{B}_1$  succeeds in the case that  $\mathcal{A}$  finds a collision against an internal application of  $f$ , whereas  $\mathcal{B}_2$  succeeds in the case that  $\mathcal{A}$  will find a collision against the enveloping function. First we prove that for both adversaries  $\delta$  is well-defined whenever  $M \neq M'$  but the messages collide under  $\text{sNI}^f$ . If  $|M| \neq |M'|$  then because  $\text{pad}_s$  includes strengthening we have trivially that  $\delta = 0$ . Otherwise  $|M| = |M'|$  and so there must exist some block  $P_i \neq P'_i$  for an  $i \in [1..m]$ . (If not, then  $M = M'$  and this violates our assumption.) Thus  $\delta$  is well defined. Now we note that since  $\delta$  is the minimal value and  $M$  and  $M'$  collide, we have that necessarily  $Y_{m-\delta} = Y'_{m'-\delta}$  and so the messages returned by either adversary must form a collision against  $f$ .

Let  $\text{Coll}_{\mathcal{A}}$  be the event that  $\mathcal{A}$  succeeds in the cr game and similarly define  $\text{Coll}_{\mathcal{B}_1}$  and  $\text{Coll}_{\mathcal{B}_2}$ . Then

$$\begin{aligned} \Pr[\text{Coll}_{\mathcal{A}}] &= \Pr[\text{Coll}_{\mathcal{A}} \wedge \delta = 0] + \Pr[\text{Coll}_{\mathcal{A}} \wedge \delta \neq 0] \\ &= \Pr[\text{Coll}_{\mathcal{B}_1}] + \Pr[\text{Coll}_{\mathcal{B}_2}]. \end{aligned} \quad (1)$$

Note that both  $\mathcal{B}_1$  and  $\mathcal{B}_2$  run in time  $t' + cT_f l$  where  $l = \max\{m, m'\}$  and  $c$  is a small constant. Letting  $\epsilon = \max_{\mathcal{B}} \{\text{Adv}_f^{\text{cr}}(\mathcal{B})\}$  for all adversaries  $\mathcal{B}$  running in time at most  $t' + cT_f l$ . Then by equation (1) we have that  $\epsilon = \epsilon'/2$ . Part (1) of the theorem follows.

The proof of part (2) is similar. Let  $\mathcal{A}$  be a  $(t', L, \epsilon')$ -cr adversary against  $\text{ESh}^f$  and we define an adversary  $\mathcal{B}$  as shown below.

Adversary  $\mathcal{B}(K)$

$\{K_i\}_1^\mu \xleftarrow{\$} (\{0, 1\}^n)^\mu; (M, M') \xleftarrow{\$} \mathcal{A}(K, \{K_i\}_1^\mu)$   
 $P_1 \dots P_m \xleftarrow{d} \text{padCS}_s(M); P'_1 \dots P'_{m'} \xleftarrow{d} \text{padCS}_s(M')$   
 $Y_0 \leftarrow Y'_0 \leftarrow IV_1$   
For  $i \in [1..m-1]$  do  $Y_i \leftarrow f_K(Y_{i-1} \oplus K_{\nu(i)}, P_i)$   
For  $i \in [1..m'-1]$  do  $Y'_i \leftarrow f_K(Y'_{i-1} \oplus K_{\nu(i)}, P'_i)$   
Let  $\delta$  be smallest value s.t.  $(Y_{m-\delta-1} \oplus K_{\nu(m-\delta)}) \parallel P_{m-\delta} \neq (Y'_{m'-\delta-1} \oplus K_{\nu(m'-\delta)}) \parallel P'_{m'-\delta}$   
If  $\delta = 0$  then Ret  $((IV_2 \oplus K_0) \parallel (Y_{m-1} \oplus K_\mu) \parallel P_m, (IV_2 \oplus K_0) \parallel (Y'_{m'-1} \oplus K_\mu) \parallel P'_{m'})$   
Ret  $((Y_{m-\delta-1} \oplus K_{\nu(m-\delta)}) \parallel P_{m-\delta}, (Y'_{m'-\delta-1} \oplus K_{\nu(m'-\delta)}) \parallel P'_{m'-\delta})$

We now show that  $\delta$  is well-defined and the message  $\mathcal{B}$  returns as a result of  $\delta$  is always a collision against  $f$  if  $\mathcal{A}$  successfully found a collision against  $\text{ESh}^f$ . If  $|M| \neq |M'|$  and  $d \geq n+64$ , then  $\text{padCS}_s$  ensures that  $\delta = 0$  and since  $\mathcal{A}$ 's messages collide the output of the envelope for these two messages is equal. If  $|M| \neq |M'|$  and  $d < n+64$ , then  $\text{padCS}_s$  ensures that  $P_m = P'_{m'} = 0^{d-n}$ . Now if  $Y_{m-1} \neq Y'_{m'-1}$  then we are done with  $\delta = 0$  because this implies also that  $Y_{m-1} \oplus K_\mu \neq Y'_{m'-1} \oplus K_\mu$  and the output of the envelope for both messages is equal. Otherwise, if  $Y_{m-1} = Y'_{m'-1}$  then  $\delta = 1$ ,

since the strengthening ensures that  $P_{m-1} \neq P'_{m'-1}$  and so  $f_K(Y_{m-2} \oplus K_{\nu(m-1)} \parallel P_{m-1}) = Y_{m-1} = Y'_{m'-1} = f_K(Y'_{m'-2} \oplus K_{\nu(m'-1)} \parallel P'_{m'-1})$ , yielding a collision against  $f$ .

If  $|M| = |M'|$ , then there must exist a  $\delta$  such that  $P_{m-\delta} \neq P'_{m'-\delta}$ , since otherwise  $M = M'$ . Because it is the least such value and  $M, M'$  collide, we have that  $Y_{m-\delta} = Y'_{m'-\delta}$  which guarantees a collision on  $f$ . Thus we have that  $\mathbf{Adv}_f^{\text{cr}}(\mathcal{B}) = \epsilon = \epsilon'$ .

Adversary  $\mathcal{B}$  can be modified easily for a reduction to sCS: just set all key masks to zero bits and do not give them to  $\mathcal{A}$  as input. The argument above holds in this case as well. Adversary  $\mathcal{B}$  runs in time  $t = t' + cT_f l$  where  $l$  is the number of blocks of the longer message output by  $\mathcal{A}$  and  $c$  is a small constant. ■

## 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. On the positive side, we have that ESh is MAC-Pr.

**Theorem 5.3 [Negative results: MD, sMD, Sh]** *If there exists a function  $f: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^{n-1}$  that is a  $(t, q, n+d, \epsilon)$ -mac, then there exists a function  $g: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$  that is a  $(t - c_1 q, q, n + d, \epsilon)$ -mac but*

(1) MD $[g]$ , sMD $[g]$  are at most  $(c_2, n - 1, 3d, 1/2)$ -mac

(2) Sh $[g]$  is at most a  $(c_3, 2(n - 1), 3d, 1/4)$ -mac

where  $c_1, c_2$ , and  $c_3$  are small constants. □

The theorem gives that MD, sMD, and Sh are not MAC-Pr: there exists a compression function  $g$  that is a good MAC but for which there exists efficient adversaries that can forge against MD $^g$ , sMD $^g$ , and Sh $^g$  with high probability. Our proof shows this by constructing a special compression function that “leaks” information about the chaining variable input. While unimportant for the unforgeability of  $g$  itself, functions built using  $g$  and one of the above transforms are susceptible to “length reduction attacks”: the MACs of longer messages give enough information to determine the MAC of a shorter message.

**Proof:** Let  $s = \lceil \log_2 n \rceil$ . We start by defining a compression function  $g$  in terms of  $f$ :

$$g_K(Y \parallel M) = \begin{cases} f_K(Y \parallel M) \parallel Y[i] & \text{if } M = \langle i \rangle_s \parallel 0^{d-s} \text{ for } i \in [1..n-1] \\ f_K(Y \parallel M) \parallel Y[n] & \text{otherwise} \end{cases} \quad (2)$$

The function  $g$  outputs a string that “leaks” information about the input  $Y$ . We show that this does not compromise the unforgeability of the compression function: as long as  $f$  is unforgeable, so too is  $g$ . Intuitively this is because an adversary against  $g$  already knows the value  $Y$ , so “leaking” it gives no information to the adversary. Formally, let  $A$  be an  $(t', q, n + d, \epsilon)$  mac-adversary that attacks the unforgeability of  $g$ . Then we build an adversary  $B$  that attacks  $f$ :  $B$  runs  $A$ , using its oracle to simulate  $g$  as per (2). When  $A$  outputs a pair  $(M, T)$ ,  $B$  outputs  $(M, T|_{n-1})$  (i.e., the tag with last bit dropped). By (2), if  $(M, T)$  is a valid forgery for  $g$  then  $(M, T|_{n-1})$  must be a valid forgery against  $f$ . Adversary  $B$  runs in time  $t' + c_1 q$  for a small constant  $c_1$ .

Now we prove part (2) of the theorem statement (part (1) will be an easy corollary). Particularly, we build a  $(c_2, 2(n - 1), 3d, 1/4)$  mac-adversary  $\mathcal{A}$  against Sh $^g$ . For notational ease let  $H_{K, \{K_i\}_1^k}(\cdot) = \text{Sh}^g((K, \{K_i\}_1^k), \cdot)$ . Figure 3 details  $\mathcal{A}$ . It first chooses a one-block message  $M$

<p>Adversary <math>\mathcal{A}^{\mathcal{O}(\cdot)}</math>:</p> <pre> 00 <math>M \leftarrow 0^d</math> 01 <b>for</b> <math>i \leftarrow 1</math> to <math>n - 1</math> <b>do</b> 02   <math>Y \leftarrow \mathcal{O}(M \parallel 10^{d-64-1} \parallel \langle d \rangle_{64} \parallel \langle i \rangle_s \parallel 0^{d-s})</math> 03   Let <math>y_i</math> be the last bit of <math>Y</math> 04 <b>for</b> <math>i \leftarrow 1</math> to <math>n - 1</math> <b>do</b> 05   <math>Z \leftarrow \mathcal{O}(\langle i \rangle_s \parallel 0^{d-s})</math> 06   Let <math>z_i</math> be the last bit of <math>Z</math> 07   <math>a, b \xleftarrow{\\$} \{0, 1\}</math> 08 <b>for</b> <math>i \leftarrow 1</math> to <math>n - 1</math> <b>do</b> 09   <math>t_i \leftarrow y_i \oplus z_i \oplus IV[i] \oplus a</math> 10 <b>Ret</b> <math>(M, t_i \parallel \dots \parallel t_{n-1} \parallel b)</math> </pre>	<p>For <math>i \in [1 .. n - 1]</math>:</p> $y_i = H_{K, \{K_i\}_1^\kappa}(M) \oplus K_0[i] \oplus K_2[n]$ $z_i = IV[i] \oplus K_0[i] \oplus K_1[n]$ $t_i = H_{K, \{K_i\}_1^t}(M) \oplus K_1[n] \oplus K_2[n] \oplus a$
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

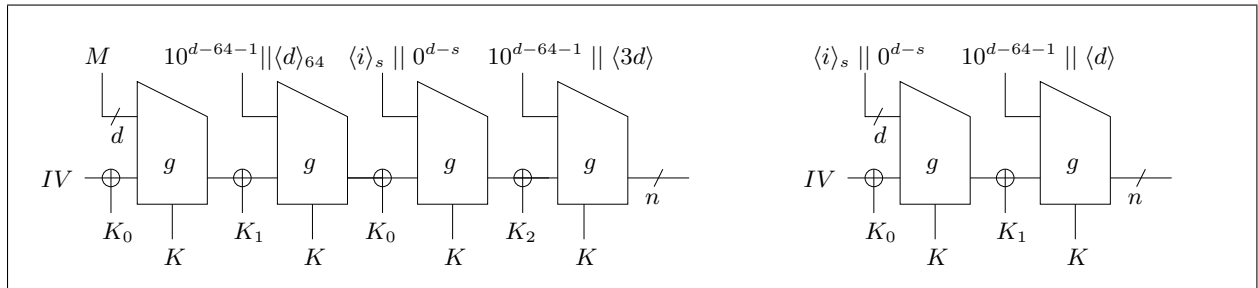


Figure 3: **(Top left)** The adversary utilized in the proof of Theorem 5.3. Recall that here the oracle  $\mathcal{O}$  implements  $H_{K, \{K_i\}_1^\kappa}(\cdot) = \text{Sh}^g((K, \{K_i\}_1^\kappa), \cdot)$ . **(Top right)** Values computed in the course of the adversary’s attack. **(Bottom)** Diagrams of the responses to the adversary’s queries.

that it will forge against. On lines 01–03 it queries its oracle (implementing  $\text{Sh}^g$ )  $n - 1$  times to gather information about an intermediate value in the calculation of a specially crafted message  $M \parallel 10^{d-64-1} \parallel \langle d \rangle_{64} \parallel \langle i \rangle_p \parallel 0^{d-s}$ . The first two blocks of the message are exactly the string  $\text{pad}_s(M)$ . The last block is used to specify which bit of the chaining variable should be “leaked”. From this  $\mathcal{A}$  learns the bit string  $H_{K, \{K_i\}_1^\kappa}(M)|_{n-1} \oplus K_0|_{n-1} \oplus K_2[n]^{n-1}$  of length  $n - 1$ . Figure 3 (bottom left) depicts the structure of the responses to these queries.

To be able to strip off the (secret) value  $K_0$ , the adversary  $\mathcal{A}$  performs a second set of  $n - 1$  queries to its oracle on the messages  $\langle i \rangle_s \parallel 0^{d-s}$  (lines 04–06). The responses allow it to learn the bit string  $IV|_{n-1} \oplus K_0|_{n-1} \oplus K_1[n]^{n-1}$  again of length  $n - 1$  bits. The bottom right diagram in Figure 3 depicts the structure of the responses. Now the adversary chooses a bit  $a$  uniformly, which is its guess of the value  $K_1[n] \oplus K_2[n]$ . This allows  $\mathcal{A}$  to construct (with probability  $1/2$ ) the bit string  $H_{K, \{K_i\}_1^\kappa}(M)|_{n-1}$ , performed by lines 08–09. It also chooses a bit  $b$  uniformly to guess the last bit of a  $H_{K, \{K_i\}_1^\kappa}(M)$  and outputs the resulting forgery. The equivalences shown in Figure 3 on the right summarize the values used by  $\mathcal{A}$ .

With probability  $1/2$  we have that  $a = K_1[n] \oplus K_2[n]$  and with probability  $1/2$  the choice of  $b$  will be correct. Thus  $\mathcal{A}$  wins with probability  $1/4$ . We see that  $\mathcal{A}$  makes  $2(n - 1)$  queries, each of length no longer than  $3d$  blocks and requires a small constant amount of time proportional to  $n$ . Part (2) of the theorem statement follows.

For part (1), we must show similar results for  $\text{MD}^g$  and  $\text{sMD}^g$ . We can modify  $\mathcal{A}$  from part (2) in the following way. Omit lines 04–06 and 08–09, change line 07 to  $b \stackrel{\$}{\leftarrow} \{0, 1\}$ , and replace line 10 with  $\text{Ret}(M, y_i \parallel \dots \parallel y_{n-1} \parallel b)$ . These changes are the result of the simpler transforms which do not use key masks. In this case  $\mathcal{A}$  makes  $n - 1$  queries of length at most  $3d$  and wins with probability  $1/2$ , justifying the bounds in part (2) of the theorem statement.  $\blacksquare$

**Theorem 5.4 [Positive results: ESh]** *Let  $f: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$  be a  $(t, q, n + d, \epsilon)$ -mac. Then  $\text{ESh}[f]$  is a  $(t - c(q' + 1)\tau(L), q', L, \epsilon')$ -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  $\{K_i\}_1^\mu \in \{0, 1\}^{\mu n}$  with  $\mu = \lceil \log_2(\tau(L) - 1) \rceil + 1$ .  $\square$

**Proof:** Let  $\{K_i\}_1^\mu \in \{0, 1\}^{\mu n}$  (these need not be public or distributed uniformly for the proof). We use the general approach to proving unforgeability for MD-style constructions due to Maurer and Sjödin [18], adapting in particular their proof that  $\text{CS}^f$  is unforgeable. Let  $\mathcal{A}$  be a  $(t', q', L, \epsilon')$  forger against  $f^{\text{ESh}}$ . Let  $\sigma$  be the total number of applications of  $f$  required to answer  $\mathcal{A}$ 's  $q'$  queries (necessarily  $\sigma \leq q'\tau(L)$ ). We build a forger  $\mathcal{B}$  against  $f$  using  $\mathcal{A}$ . First,  $\mathcal{B}$  picks a *forging strategy* randomly from a set of potential forging strategies  $\mathcal{FS}$  and then it runs  $\mathcal{A}$ , proceeding according to the strategy. A forging strategy will specify how  $\mathcal{B}$  is to use  $\mathcal{A}$  to forge against  $f$ . For all the strategies  $\mathcal{B}$  simulates  $\mathcal{A}$ 's  $\text{ESh}^f$  oracle using its own  $f$  oracle, in the natural way. Upon  $\mathcal{A}$  halting with output a forgery message and tag (if  $\mathcal{B}$  had not already stopped the simulation),  $\mathcal{B}$  computes  $\text{ESh}^f$  on the forgery message using its oracle.

We will show that the set of forging strategies we specify in a moment is complete, which means that if  $\mathcal{A}$  succeeds in forging then at least one of the strategies in  $\mathcal{FS}$  is successful at extracting a forgery against  $f$ . Adversary  $\mathcal{B}$ 's advantage will then simply be equal to  $\mathcal{A}$ 's advantage divided by the size of  $\mathcal{FS}$ . We define the set of potential forging strategies as

$$\begin{aligned} \mathcal{FS} = & \{(j, IV_1) : j \in [1.. \sigma]\} \cup \{(j, IV_2) : j \in [1.. \sigma]\} \cup \\ & \{(j, i) : j \in [1.. \sigma] \wedge 1 \leq i < j\} \cup \{(\sigma, \text{T})\} \end{aligned}$$

A strategy  $(j, z)$  specifies that  $\mathcal{B}$  should run  $\mathcal{A}$ , simulating its  $\text{ESh}^f$  oracle as described above. The number  $j$  tells  $\mathcal{B}$  when to stop the simulation: right before the  $j^{\text{th}}$  query to its  $f$  oracle. The value  $z$  specifies a forgery value to output right after  $\mathcal{B}$  stops the simulation. Let  $N^j \in \{0, 1\}^{n+d}$  be the string that would have formed  $\mathcal{B}$ 's  $j^{\text{th}}$  query. Let  $K^j$  be the key mask associated with the  $j^{\text{th}}$  query by  $\mathcal{B}$ . Particularly,  $K^j = K_\mu$  or  $K^j = K_{\nu(c)}$  where  $c$  is one more than the number of queries used thus far to simulate  $\mathcal{A}$ 's current query (i.e., the query causing  $\mathcal{B}$ 's  $j^{\text{th}}$  query). Then there are four types of values for  $z$  in  $\mathcal{FS}$ :

- If  $z = IV_1$ , then  $\mathcal{B}$  returns  $(N^j, IV_1 \oplus K^{j+1} \oplus K_0)$ . This corresponds to  $\mathcal{B}$  predicting that  $\mathcal{A}$  causes some chaining variable to equal  $IV_1$  (after accounting for the key masks).
- If  $z = IV_2$  then  $\mathcal{B}$  returns  $(N^j, IV_2 \oplus K^{j+1} \oplus K_0)$ . This corresponds to  $\mathcal{B}$  predicting that  $\mathcal{A}$  causes some chaining variable to equal  $IV_2$  (after accounting for the key masks).
- If  $z = i$  for some  $i < j$ , then  $\mathcal{B}$  returns  $(N^j, Y^i \oplus K^{i+1} \oplus K^{j+1})$  where  $Y^i$  is the response from  $\mathcal{B}$ 's  $i^{\text{th}}$  query. This corresponds to  $\mathcal{B}$  predicting that  $\mathcal{A}$  causes the outputs of two internal chaining variables to collide (after accounting for the key masks).
- If  $z = \text{T}$  (here  $\text{T}$  is just a flag), then  $\mathcal{B}$  returns  $(N^\sigma, T)$  where  $T$  is the forgery tag output by  $\mathcal{A}$ . This is referred to as the naive strategy in [18] and corresponds to  $\mathcal{B}$  predicting that  $\mathcal{A}$  can correctly predict the output  $T$  of  $\text{ESh}^f$  on the forgery message.

We now show that  $\mathcal{FS}$  is complete: if the coins given to  $\mathcal{A}$  would result in a forgery for a given choice of  $K$  then at least one of the strategies in  $\mathcal{S}$  is successful. We do this via case analysis based on the four types of strategies. Assume that strategy  $(\sigma, \mathbf{T})$  is not successful (otherwise the claim is proven). The only way this strategy does not succeed is if there existed a previous query by  $\mathcal{B}$  on the same message, i.e there exists a query index  $i < \sigma$  such that  $N^i = N^\sigma$ . Let  $i$  be the earliest such query. Now we show that this implies that one of the other strategies succeeds. We do so via case analysis regarding the query  $i$ .

First consider the case in which query  $i$  was queried while simulating the internal iteration of  $\text{ESh}^f$  (as opposed to the envelope). Then this means that the first  $n$  bits of  $N^i$  must equal  $IV_2 \oplus K_0$ . If  $N^i$  was queried as the first application of  $f$  for one of  $\mathcal{A}$ 's queries, then this would mean that the low  $n$  bits of  $N^i$  are equal to  $IV_1 \oplus K_0$  which can't equal  $IV_2 \oplus K_0$  since  $IV_1 \neq IV_2$ . Therefore, the low  $n$  bits of  $N^i$  are equal to  $f_K(N^{i-1}) \oplus K^i$ . But for  $N^i = N^\sigma$  we then have that  $f_K(N^{i-1}) = IV_2 \oplus K^i \oplus K_0$ . This implies that the  $(i-1, IV_2)$  strategy succeeds.

The other case is that query  $i$  was made to compute the envelope of  $\text{ESh}^f$  for some query by  $\mathcal{A}$ . This means that  $N^i$  is of the form  $(IV_2 \oplus K_0) \parallel (Y^{i-1} \oplus K_\mu) \oplus P^i$  where  $P^i$  are message bits specified by  $\mathcal{A}$ . Then for  $N^i = N^\sigma$  to hold,  $Y^{i-1} \oplus K_\mu = Y^{\sigma-1} \oplus K_\mu$  and so  $Y^{i-1}$  must equal  $Y^{\sigma-1}$ . (Recall that  $f^{\text{ESh}}$  specifies that the mask for the envelope is always fixed to  $K_\mu$ .) Let  $P_1 \cdots P_m \in D^\circ$  be the query by  $\mathcal{A}$  (after padding) that resulted in query  $i$  by  $\mathcal{B}$ . Let  $P'_1 \cdots P'_{m'} \in D^\circ$  be the forgery message output by  $\mathcal{A}$  (after padding). Define the intermediate chaining variables  $Y_i = f_K(Y_{i-1} \oplus K_{\nu(i)} \parallel M_i)$  for  $1 \leq i \leq m-1$  and  $Y'_i = f_K(Y_{i-1} \oplus K_{\nu(i)} \parallel P_i)$  for  $1 \leq i \leq m'-1$  where  $Y_0 = Y'_0 = IV_1$ . Then we have that  $Y^{i-1} = Y_{m-1}$  and  $Y^\sigma = Y'_{m'-1}$ . Let  $\delta$  such that  $0 < \delta < \min\{m, m'\}$  be the least value such that  $Y_{m-\delta-1} \parallel P_{m-\delta} \neq Y'_{m'-\delta-1} \parallel P'_{m'-\delta}$ . We have three potential cases:

- $0 < \delta < \min\{m, m'\}$ : In this case we have that the  $(j-\delta, i-\delta)$  strategy must succeed.
- $\delta$  is undefined and  $m \neq m'$ : Without loss of generality assume that  $m < m'$  (the argument for the other direction is symmetric). Because  $\delta$  is undefined we have that  $(IV_1 \oplus K_0) \parallel P_1 = (Y_{m'-m} \oplus K_{\nu(m'-m)}) \parallel P'_{m'-m}$ , which means that  $Y_{m'-m} = IV_1 \oplus K_{\nu(m'-m)} \oplus K_0$ . This implies that the  $(\sigma-m, IV_1)$  strategy succeeds.
- $\delta$  is undefined and  $m = m'$ : This case cannot occur because it implies that  $P_1 \cdots P_m = P'_1 \cdots P'_{m'}$ , which means  $\mathcal{A}$  did not output a valid forgery.

So in all cases at least one of  $\mathcal{B}$ 's strategies succeeds, and so the probability of  $\mathcal{B}$ 's success is simply equal to  $\epsilon'$  divided by the total number of strategies, which we now count. The first and second kinds of strategies contribute  $\sigma$  distinct tuples each. The third kind of strategy has at most  $(\sigma^2 - \sigma)/2$  total tuples. The final type of strategy contributes a single tuple. So  $|\mathcal{FS}| = (\sigma^2 - \sigma)/2 + 2\sigma + 1$  and therefore we have that  $\mathcal{B}$ 's advantage is at least

$$\epsilon = \frac{\epsilon'}{0.5\sigma^2 - 0.5\sigma + 2\sigma + 1}$$

and solving for  $\epsilon'$  we have that

$$\epsilon' = \left( \frac{\sigma^2 + 3\sigma}{2} + 1 \right) \epsilon.$$

In the worst case all of  $\mathcal{A}$ 's queries require  $\tau(L)$  queries to  $f$  by  $\mathcal{B}$ . This means that  $\sigma = q = q'\tau(L) + \tau(L) - 1$  where we add in the extra  $\tau(L) - 1$  applications of  $f$  used for the forgery. Solving for  $q'$  we have that  $q' = (q - \tau(L) + 1)/\tau(L)$  as specified in the theorem statement. Adversary  $\mathcal{B}$  runs  $\mathcal{A}$  and needs a small constant amount of overhead for each block of message output by  $\mathcal{A}$  (the queries and the forgery). Thus  $t = t' + c(q' + 1)\tau(L)$ . ■

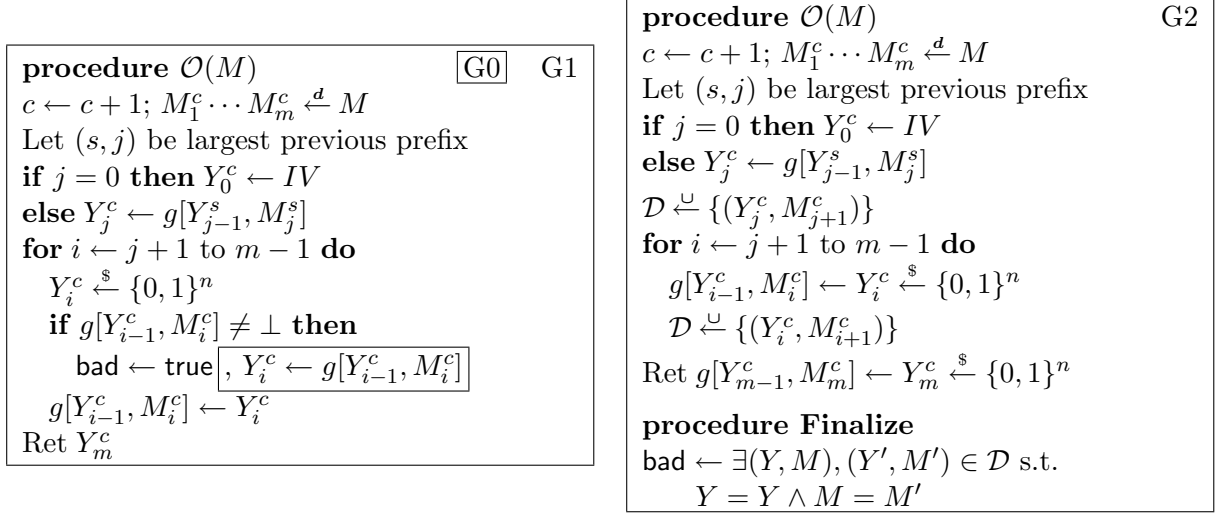


Figure 4: Games used in proof that MD is PRF-Pr. Initially the table  $g$  is everywhere set to  $\perp$  and  $c = 0$ . The “largest previous prefix” is shorthand for specifying the largest values  $(s, j)$  such that there exists a query  $M^s$  for which  $M_i^s = M_i^c$  for  $1 \leq i \leq j$ .

### 5.3 Pseudorandom Function Preservation

In the non-dedicated-key setting, building PRF preserving transforms is non-trivial and the proofs of security can be quite complex [3, 4, 2]. In stark contrast to this, we show that *all* of the dedicated-key transforms considered here are PRF-Pr, and the proof establishing this is relatively straightforward. We note that the main difference between the two settings is that length-extension attacks, possible in the non-dedicated-key setting, are no longer possible here because the adversary can not compute the (secretly) keyed compression function on its own.

**Theorem 5.5 [Positive results: MD, sMD, Pre, Sh, sNI, NI, CS, sCS, ESh]** *Let  $f: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$  be a  $(t, q, n + d, \epsilon)$ -prf. Then  $\mathbb{T}[f]$  is a  $(t', q', L, \epsilon')$ -prf where for*

- (1)  $\mathbb{T} \in \{\text{MD, sMD, Pre, Sh, CS, sCS, ESh}\}$ ,  $t' = t - cq\tau(L)$ ,  $q' = q/\tau(L)$ ,  $\epsilon' = \epsilon + q^2\tau(L)^2/2^n$ ,
- (2)  $\mathbb{T} \in \{\text{sNI, NI}\}$ ,  $t' = t - cq\tau(L)$ ,  $q' = q/\tau(L)$ ,  $\epsilon' = 2\epsilon + q^2(\tau(L) - 1)^2/2^n$

where  $c$  is a small constant and  $\{K_i\}_1^\mu \in \{0, 1\}^{\mu n}$  for  $\mu = \lceil \log_2(\tau(L) - 1) \rceil + 1$ .  $\square$

**Proof:** We start by proving that the MD iteration  $f^+$  is a good PRF as long as  $f$  is also. This immediately implies the part (1) results for MD, sMD, and Pre; the others are straightforward corollaries. Let  $\mathcal{A}$  be a  $(t', q', L, \epsilon')$ -prf-adversary that attempts to distinguish between  $f^+(K, \cdot)$  for  $K \stackrel{s}{\leftarrow} \{0, 1\}^k$  and  $\rho \stackrel{s}{\leftarrow} \text{Func}(D^+, n)$ . Let  $\sigma$  be the max number of message blocks queried by  $\mathcal{A}$ . Let  $\text{Adv}(\mathcal{A}^{(\cdot)_1}, \mathcal{A}^{(\cdot)_2}) \equiv \Pr[\mathcal{A}^{(\cdot)_1} \Rightarrow 1] - \Pr[\mathcal{A}^{(\cdot)_2} \Rightarrow 1]$ . We utilize a hybrid argument, bounding

$$\text{Adv}(\mathcal{A}^{f^+(K, \cdot)}, \mathcal{A}^{\rho^{(\cdot)}}) = \text{Adv}(\mathcal{A}^{f^+(K, \cdot)}, \mathcal{A}^{g^+(\cdot)}) + \text{Adv}(\mathcal{A}^{g^+(\cdot)}, \mathcal{A}^{\rho^{(\cdot)}})$$

where  $K \stackrel{s}{\leftarrow} \{0, 1\}^k$ ,  $g \stackrel{s}{\leftarrow} \text{Func}(n + d, n)$ , and  $\rho \stackrel{s}{\leftarrow} \text{Func}(D^+, n)$ . Here  $g^+(M)$  is just the MD iteration but replacing  $f_K$  with a truly random compression function  $g$ . We can use a standard argument to bound the first term by  $\epsilon$ , the advantage of a prf-adversary attempting to distinguish between  $f$  and  $g$  that runs in time  $t = t' + cq\tau(L)$  and using  $q = q'\tau(L)$  queries

We bound the difference between the second two terms with a simple game-playing argument [10]. Figure 4 shows two games, G0 and G1. Game G0 (boxed statement included) implements an oracle



that exactly simulates the construction  $g^+$  where  $g \stackrel{\$}{\leftarrow} \text{Func}(n+d, n)$ . Game G1 (boxed statement removed) replies with a random sequence of  $n$  bits for every query, thus simulating exactly a random function  $\rho \stackrel{\$}{\leftarrow} \text{Func}(D^+, n)$ . The two games are identical-until-*bad* and so by the above reasoning and the fundamental lemma of game playing [10] we have that

$$\mathbf{Adv}(\mathcal{A}^{g^+(\cdot)}, \mathcal{A}^{\rho(\cdot)}) = \mathbf{Adv}(\mathcal{A}^{\text{G0}}, \mathcal{A}^{\text{G1}}) \leq \Pr[\mathcal{A}^{\text{G1}} \text{ sets bad}].$$

We slightly modify G1 to yield game G2: instead of setting *bad* up-front, we collect all the domain points of  $g$  in a multiset  $\mathcal{D}$ . If at the end of the game  $\mathcal{D}$  contains two identical pairs then we set *bad*. Since everytime *bad* was set in G1 a duplicate pair will be placed in  $\mathcal{D}$  in G2 we have that  $\Pr[\mathcal{A}^{\text{G1}} \text{ sets bad}] = \Pr[\mathcal{A}^{\text{G2}} \text{ sets bad}]$ . We can now bound the probability of *bad* being set in G2 as follows. We have that during finalization  $|\mathcal{D}| \leq q\sigma$ . For each pair in  $\mathcal{D}$ , the first element is chosen independently from all others first elements. Thus we have that

$$\Pr[\mathcal{A}^{\text{G2}} \text{ sets bad}] \leq \Pr[\exists(Y, M), (Y', M') \in \mathcal{D} \text{ s.t. } Y = Y'] = \binom{q\sigma}{2} \frac{1}{2^n} \leq \frac{q^2\sigma^2}{2^n}.$$

Substituting  $\tau(L)$  appropriately for  $\sigma$  and combining with the other portion of the hybrid gives  $\epsilon'$  as specified in the theorem statement.

The proof can be straightforwardly modified to handle Sh, CS, sCS, and ESh. We omit the details, but point out that, in fact, the result holds even if the key masks are made public (hence the quantification over any possible key masks made in the theorem statement). To prove part (2), simply expand the hybrid argument to account for two different uses of  $f$ . Then observe that using the same (truly random) compression function in the games can only lead to a looser bound. ■

## 5.4 Pseudorandom Oracle Preservation

Establishing that a transform is PRO-Pr ensures that the constructed hash function “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  $f: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$  as a family of random oracles, one for each key in  $\{0, 1\}^k$ . However, since we only ever use one or two keys from  $\{0, 1\}^k$ , we will (without loss of generality) simply utilize two separate random oracles  $f = \text{RF}_{n+d,n}$  and  $g = \text{RF}_{n+d,n}$  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  $\text{MD}^f$  and  $\text{sMD}^f$  are not PROs follows from simple length-extension attacks (see [11]) and the security of sCS is implied by the security of EMD [5].

We point out that  $\text{Sh}^f$  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  $\text{ESh}^f$  is a PRO.

**Theorem 5.6 [Negative result: Sh]** *Let  $f = \text{RF}_{n+d,n}$  be a random oracle. Then there exists an  $(c, t_S, 1, 2, 2d, 1 - 2^{-n})$ -pro adversary  $\mathcal{A}$  against  $\text{Sh}^f$  for any simulator  $\mathcal{S}$  with arbitrary running time  $t_S$ . The running time of  $\mathcal{A}$  is a small constant  $c$ . □*

**Proof:** Let  $\mathcal{A}$  work as follows. Recall that it has two oracles  $\mathcal{O}_1$  and  $\mathcal{O}_2$  and is run on input  $K, \{K_i\}_1^k$ . It chooses random strings  $M_1 \in \{0, 1\}^{d-65}$  and  $M_2 \in \{0, 1\}^{d-65}$ . It queries  $\mathcal{O}(M_1)$  to get response  $Y$

and  $\mathcal{O}_1(M_1 \parallel 1 \parallel \langle d \rangle_{65} \parallel M_2)$  to get response  $Z$ . It then queries  $\mathcal{O}_2((Y \oplus K_1) \parallel M_2 \parallel 1 \parallel \langle d \rangle_{65})$ , receiving  $Z'$ . If  $Z = Z'$  then it outputs 1 (guessing it is in the world where  $\mathcal{O}_1$  is  $\text{Sh}^f$ ) and otherwise outputs 0. It is easy to verify that  $\Pr[\mathcal{A}^{\text{Sh}^f((K, \{K_i\}_1^\kappa), \cdot), f(\cdot)} \Rightarrow 1] = 1$  where the probability is taken over random choice of  $f$  and  $K, \{K_i\}_1^\kappa$  and the coins used by  $\mathcal{A}$ . On the other hand  $\Pr[\mathcal{A}^{\mathcal{F}(\cdot), \mathcal{S}^{\mathcal{F}}((K, \{K_i\}_1^\kappa), \cdot)} \Rightarrow 1] = 2^{-\min\{n, d-65\}}$  where the probability is taken over random choice of  $\mathcal{F}$  and  $K, \{K_i\}_1^\kappa$  and the coins used by  $\mathcal{A}$  and  $\mathcal{S}$ . This is so because the simulator is only queried once, at which point it has no information about  $M_1$ . It can either guess  $M_1$  or the appropriate output  $Z'$ . (In the case that  $n \geq d - 65$  we can increase the advantage back to  $1 - 2^{-n}$  by utilizing more message blocks.) ■

**Theorem 5.7 [Positive result: ESh]** *Let  $f = \text{RF}_{n+d,n}$  be a random oracle and let  $\{K_i\}_1^\mu \in \{0, 1\}^{\mu n}$  and let  $IV_1, IV_2 \in \{0, 1\}^n$  with  $IV_1 \neq IV_2$ . Then  $\text{ESh}^f$  is a  $(t_{\mathcal{A}}, t_{\mathcal{S}}, q_1, (q_2 + q_3), L, \epsilon)$ -pro where*

$$\epsilon \leq \frac{l^2 q_1^2 + (l q_1 + q_2)(q_2 + q_3)}{2^n} + \frac{l q_1 + q_2}{2^n}$$

for  $l = \tau(L)$  and  $L$  being the maximal message length queried. Here  $q_2$  is the number of queries to the  $f$  oracle with low  $n$  bits not equal to  $IV_2 \oplus K_0$  and  $q_3$  is the number of queries to  $f$  with low  $n$  bits equal to  $IV_2 \oplus K_0$ . The running time  $t_{\mathcal{A}}$  is arbitrary while  $t_{\mathcal{S}} = \mathcal{O}(q_2^2 + q_2 q_3)$ . □

**Proof:** We adapt the proof that EMD is PRO-Pr from [5] to prove a more general result, that  $f^{\text{ESh}}$ , the iteration function underlying  $\text{ESh}^f$ , is a PRO. The core of the proof is captured by two lemmas, one showing that we can bound the advantage of an adversary against  $f^{\text{ESh}}$  by the advantage of an adversary against  $gf^{\text{ESh}}$ . This is the  $f^{\text{ESh}}$  iteration except with the last application of  $f$  (the envelope) replaced by a distinct FIL random oracle  $g$ . The second lemma bounds the ability of any adversary against  $gf^{\text{ESh}}$ .

First we detail the two simulators  $SA$  and  $SB$  used in the proof, shown in Figure 5. Both simulators have access to all the key masks  $\{K_i\}_1^\mu$ , which can be any constant values. The first simulates a single random oracle  $f$  while the second simulates two random oracles  $f$  and  $g$ . We concentrate on explaining the simulator  $SB$ . It internally builds a tree data structure that serves to correlate queries from the adversary. The tree initially has one root node labeled  $IV_1$ . Now, upon receiving an  $f$ -query on  $X$  the simulator parses  $X$  into  $V$  and  $U$  of lengths  $n$  and  $d$  bits. It then checks to see if  $V$  is equal to the  $IV_1$  xor'd with the key mask  $K_0$ . If so, this query is believed to correspond to the first application of  $f$  in handling a message (whose first message block is  $U$ ). The simulator therefore adds a new node to the tree labeled with the randomly chosen chaining variable  $Y$ , and adds an edge between the (root) node labeled  $IV_1$  and the new node. The edge is labeled  $U$ . The operation of adding such a node and edge to the tree is notated by  $\text{NEWNODE}(U) \leftarrow Y$ .

If  $V \neq IV_1 \oplus K_0$ , then the simulator checks the state of the tree for nodes that  $V$  should be associated with. The operation  $M_1 \cdots M_i \leftarrow \text{GETNODE1}(V)$  performs this, as follows. For each node with label  $N$  at depth  $i$  in the tree, the simulator checks if  $V = N \oplus K_{\nu(i)}$ . If so, then the simulator sets  $M_1 \cdots M_i$  to be the concatenation of the labels of the edges on the path from the root to this node. (If no such node is found, then the empty string is returned and the predicate evaluates to false.) Next the simulator adds a new node labeled with the randomly chosen chaining variable  $Y$  and adds an edge from the node just found (i.e., the one located at the end of the path labeled by  $M_1 \cdots M_i$ ) with label  $U$ . This operation is represented by  $\text{NEWNODE}(M_1 \cdots M_i U) \leftarrow Y$ . In this way the simulator builds the tree that tracks all queries.

For  $g$ -queries, the simulator parses a query  $X$  into  $W, V$ , and  $U$  of lengths  $n, n$ , and  $d - n$ . If  $W = IV_2$  then the simulator looks for a node in a similar same manner as it did during  $f$ -queries

<pre> ON QUERY <math>SB_f(X)</math>: <math>Y \stackrel{\\$}{\leftarrow} \{0, 1\}^n</math> Parse <math>X</math> into <math>V \parallel U</math> s.t. <math> U  = d,  V  = n</math> <b>if</b> <math>V = IV_1 \oplus K_0</math> <b>then</b> <math>\text{NEWNODE}(U) \leftarrow Y</math> <b>if</b> <math>M_1 \cdots M_i \leftarrow \text{GETNODE1}(V)</math> <b>then</b>   <math>\text{NEWNODE}(M_1 \cdots M_i U) \leftarrow Y</math> Ret <math>Y</math>  ON QUERY <math>SB_g(X)</math>: Parse <math>X</math> into <math>W \parallel V \parallel U</math> s.t. <math> V  = n,  U  = d - n,  W  = n</math> <b>if</b> <math>W = IV_2 \oplus K_0</math> <b>and</b>   <math>M_1 \cdots M_i \leftarrow \text{GETNODE2}(V)</math> <b>then</b>     Ret <math>\mathcal{F}(M_1 \cdots M_i U)</math> Ret <math>Y \stackrel{\\$}{\leftarrow} \{0, 1\}^n</math> </pre>	<pre> ON QUERY <math>SA(X)</math>: <math>Y \stackrel{\\$}{\leftarrow} \{0, 1\}^n</math> Parse <math>X</math> into <math>W \parallel V \parallel U</math> s.t. <math> W  = n,  V  = n,  U  = d - n</math> <b>if</b> <math>W = IV_2 \oplus K_0</math> <b>then</b>   <b>if</b> <math>M_1 \cdots M_i \leftarrow \text{GETNODE2}(V)</math> <b>then</b>     Ret <math>\mathcal{F}(M_1 \cdots M_i U)</math>   <b>else</b>     Ret <math>Y</math> Parse <math>X</math> into <math>U \parallel V</math> s.t. <math> U  = d,  V  = n</math> <b>if</b> <math>V = IV_1 \oplus K_0</math> <b>then</b> <math>\text{NEWNODE}(U) \leftarrow Y</math> <b>if</b> <math>M_1 \cdots M_i \leftarrow \text{GETNODE1}(V)</math> <b>then</b>   <math>\text{NEWNODE}(M_1 \cdots M_i U) \leftarrow Y</math> Ret <math>Y</math> </pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 5: Pseudocode for simulators  $SB$  and  $SA$  utilized in the proof of Theorem 5.7.

by running  $M_1 \cdots M_i \leftarrow \text{GETNODE2}(V)$ . The only difference is that the comparison is of the form  $N = V \oplus K_\mu$  (i.e., always the same mask is used regardless of the depth). If a matching node is found, then the simulator queries its VIL random oracle on  $M_1 \cdots M_i U$  and returns that value, thus programming the  $g$  FIL random oracle to match the VIL RO's output for the message  $M_1 \cdots M_i U$ .

Intuitively, as long as there are no collisions in the choices of chaining variables  $Y$  (after accounting for the masks) and the adversary can not predict any of these choices, then the simulator should be able to always program its responses to  $g$ -queries in order to match up with queries  $\mathcal{A}$  makes to the VIL random oracle. The simulator  $SA$  is just like  $SB$ , except that it distinguishes between “ $g$ -queries” and “ $f$ -queries” based on the first  $n$  bits of the query: if it's equal to  $IV_2 \oplus K_0$  then it's a “ $g$ -query” (corresponding to the envelope), otherwise it's an “ $f$ -query” (corresponding to an internal application of the compression function). Simulator  $SA$  runs in time  $\mathcal{O}(q_2^2 + q_2 q_3)$  where  $q_2$  is the number of queries it receives with low  $n$  bits not equal to  $IV_2 \oplus K_0$  and  $q_3$  is the number of queries it receives with first  $n$  bits equal to  $IV_2 \oplus K_0$ .

The next lemma captures the fact that any adversary attacking  $f^{\text{ESh}}$  can be turned into an adversary that attacks  $gf^{\text{ESh}}$ , which is the same as  $f^{\text{ESh}}$  but with the envelope using a separate FIL random oracle  $g$ .

**Lemma 5.8** *Let  $\mathcal{A}$  be an adversary making at most  $q_1$  queries to its first oracle of size at most  $ld + d - n$  bits. Then there exists an adversary  $\mathcal{B}$  such that*

$$\Pr \left[ \mathcal{A}^{f^{\text{ESh}}, f} \Rightarrow 1 \right] - \Pr \left[ \mathcal{A}^{\mathcal{F}, SA} \Rightarrow 1 \right] \leq \Pr \left[ \mathcal{B}^{gf^{\text{ESh}}, f, g} \Rightarrow 1 \right] - \Pr \left[ \mathcal{B}^{\mathcal{F}, SB_f, SB_g} \Rightarrow 1 \right] + \frac{lq_1 + q_2}{2^n}. \quad \square$$

We can easily adapt the proof of Theorem 5.2 from [5] to get this result. Briefly, the proof shows that the probability of the simulator choosing a range point for  $f$  equal to  $IV_2$  is low. Here we must account for the mask values, but xor'ing in constants does not change the distributions involved. We omit the details.

The next lemma bounds an adversary  $\mathcal{B}$ 's ability to differentiate  $gf^{\text{ESh}}$  from  $\mathcal{F}$ .

**Lemma 5.9** *Let  $\mathcal{B}$  be an adversary making at most  $q_1$ ,  $q_2$ , and  $q_3$  queries to its first, second, and third oracles with maximal query length of size at most  $ld + d - n$  bits. Then*

$$\Pr \left[ \mathcal{B}^{g^{f^{\text{ESh}}, f, g}} \Rightarrow 1 \right] - \Pr \left[ \mathcal{B}^{\mathcal{F}, SB_f, SB_g} \Rightarrow 1 \right] \leq \frac{l^2 q_1^2 + (l q_1 + q_2)(q_2 + q_3)}{2^n}. \quad \square$$

We can straightforwardly modify the proof of the similar Lemma 5.1 from [5] by just accounting for the mask values. We therefore omit the (lengthy) details. Combining the two lemmas gives the result.  $\blacksquare$

## 5.5 Target Collision Resistance Preservation

Universal one-way hash functions (UOWHF) were first introduced by Naor and Reingold [21]; we use the term target collision resistance (TCR), following [9]. Known transforms that preserve TCR require a logarithmic (in the maximum message length) amount of key material. Mironov has given strong evidence that one cannot do better for MD-style transforms [20]. Furthermore, any CR function is also TCR [26], and so one might simply stop with a dedicated-key transform that preserves the four properties already considered. Still, target-collision resistant functions are useful in some settings [9] and achieving it just on the basis of a TCR compression function yields stronger security guarantees. 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, we show that ESh is TCR-Pr, using the approach due to Mironov [20].

**Theorem 5.10 [Negative results: Pre, sNI, NI, CS, sCS]** *If there exists a function  $f: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^{n-1}$  that is  $(t, n + d, \epsilon)$ -tcr, then there exists a function  $g: \{0, 1\}^k \times \{0, 1\}^{(n+d)} \rightarrow \{0, 1\}^n$  that is  $(t - c_1, n + d, \epsilon + 2^{-k+1})$ -tcr but Pre[ $g$ ], sNI[ $g$ ], NI[ $g$ ], CS[ $g$ ], and sCS[ $g$ ] are at most  $(c_2, 3(d - k) - 1, 1 - 2^{-k})$ -tcr where  $c_1, c_2$  are small constants.  $\square$*

**Proof:** We utilize the counter-example from Proposition 5.1 in [9]. Let  $f: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$  where  $m > k$  be  $(t, \epsilon)$ -tcr. Then define  $g: \{0, 1\}^k \times \{0, 1\}^{(n+k)+d'} \rightarrow \{0, 1\}^{n+k}$  where  $d' = d - k$  as follows for any  $X \in \{0, 1\}^n$ ,  $Y \in \{0, 1\}^k$ , and  $Z \in \{0, 1\}^{d'}$

$$g_K(X \parallel Y \parallel Z) = \begin{cases} f_K(X \parallel Y \parallel Z) \parallel K & \text{if } y \neq K \\ 1^n \parallel 1^k & \text{if } y = K \end{cases}.$$

Bellare and Rogaway proved that  $g$  is  $(t - \Theta(k + m), \epsilon + 2^{-k+1})$ -tcr, and so we refer the reader to [9] for the analysis. Now we point out that T[ $g$ ] is insecure for  $\mathsf{T} \in \{\text{Pre}, \text{sNI}, \text{NI}, \text{CS}, \text{sCS}\}$  by showing adversaries  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$  for each.

Let  $\mathcal{A}_1$  return  $0^{d'-1} \parallel 0^{d'-2}$  and  $\mathcal{A}_2$  return  $1^{d'-1} \parallel 0^{d'-2}$ . If  $K \neq IV|_k$ , then

$$\begin{aligned} \text{Pre}_K^H(0^{d'-1} \parallel 0^{d'-2}) &= g^{\text{Pre}}(K, 0^{d'} \parallel 10^{d'-1}1) \\ &= g_K(g_K(IV \parallel 0^{d'}) \parallel 10^{d'-1}1) \\ &= g_K(f_K(IV \parallel 0^{d'}) \parallel K \parallel 10^{d'-1}1) \\ &= 1^c \parallel 1^k \end{aligned}$$

and

$$\begin{aligned}
\text{Pre}_K^H(1^{d'-1} \parallel 0^{d'-2}) &= g^{\text{Pre}}(K, 01^{d'-1} \parallel 10^{d'-1}1) \\
&= g_K(g_K(IV \parallel 01^{d'-1}) \parallel 10^{d'-1}1) \\
&= g_K(f_K(IV \parallel 01^{d'-1}) \parallel K \parallel 10^{d'-1}1) \\
&= 1^c \parallel 1^k.
\end{aligned}$$

Since  $K \neq IV|_k$  with probability  $1 - 2^{-k}$ , we have that  $\mathcal{A}$  wins with probability  $1 - 2^{-k}$ .

For sNI have  $\mathcal{A}_1$  return  $0^{d'} \parallel 0^{d'} \parallel 0^{d'-65}$  and  $\mathcal{A}_2$  return  $1^{d'} \parallel 0^{d'} \parallel 0^{d'-65}$ . For NI have  $\mathcal{A}_1$  return  $0^{d'} \parallel 0^{d'} \parallel 0^{d'-1}$  and  $\mathcal{A}_2$  return  $1^{d'} \parallel 0^{d'} \parallel 0^{d'-1}$ . For CS have  $\mathcal{A}_1$  return  $0^{d'} \parallel 0^{d'} \parallel 0^{d'-(n+k)-1}$  and  $\mathcal{A}_2$  return  $1^{d'} \parallel 0^{d'} \parallel 0^{d'-(n+k)}$ . For sCS have  $\mathcal{A}_1$  return  $0^{d'} \parallel 0^{d'} \parallel 0^{d'-(n+k)-65}$  and  $\mathcal{A}_2$  return  $1^{d'} \parallel 0^{d'} \parallel 0^{d'-(n+k)}$ . In each case a collision on the internal MD chain occurs before the enveloping application of  $g$  and so collisions are guaranteed with the same probability as for Pre. ■

**Theorem 5.11 [Positive result: ESh]** *Let  $f: \{0, 1\}^k \times \{0, 1\}^{n+d} \rightarrow \{0, 1\}^n$  be  $(t, n + d, \epsilon)$ -tcr. Then  $\text{ESh}[f]$  is  $(t - cT_f\tau(L), L, \epsilon\tau(L))$ -tcr for a small constant  $c$  and where  $T_f$  is the time to compute  $f$ . □*

**Proof:** One might attempt a black-box reduction to the TCR security of the Sh transform, however the proof from [20] is only given for FIL adversaries and so although we utilize Mironov’s techniques (in particular, his key reconstruction algorithm), we include a full proof here. Let  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$  be a tcr-adversary against  $\text{ESh}^f$ . The adversary  $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$  specified in Figure 6 utilizes  $\mathcal{A}$  as a subroutine. In the first stage  $\mathcal{B}_1$  runs  $\mathcal{A}_1$ , receiving its target message  $M$ . Adversary  $\mathcal{B}_1$  then chooses a value  $\delta$  at random, which it uses to “guess” which compression function input will be involved in a collision against  $f$  if  $\mathcal{A}$  succeeds in finding a collision against  $\text{ESh}^f$ . (Note that  $\delta$  is chosen so that  $l - \delta$  is the index of the guessed block.) Then  $\mathcal{B}_1$  sets its target message to include a randomly chosen value  $Y$  and the message block corresponding to the guess  $\delta$  (and, if  $\delta = 0$ , then  $IV_2$  is included appropriately). The random value  $Y$  represents the chaining variable value output after the  $(l - \delta - 1)^{\text{th}}$  iteration of the compression function xor’d with  $K_{\nu(l-\delta)}$ . (This gets around the fact that  $\mathcal{B}_1$  can not yet compute  $Y_{l-\delta-1}$  because it does not have the key  $K$ .)

In the second stage  $\mathcal{B}_2$  now has the key  $K$  and must generate the key masks. If  $\delta = 0$  ( $\mathcal{B}$ ’s “guess” is that a collision will be found against the envelope), then  $\mathcal{B}$  simply chooses the first  $\mu - 1$  masks at random. The last mask  $K_\mu$  (used for the chaining variable input of the envelope) is chosen such that  $K_\mu \oplus Y \oplus Y_{l-1}$  where  $Y_{l-1}$  is the chaining variable output after applying the internal Shoup iteration to  $\mathcal{A}$ ’s target message. If  $\delta \neq 0$ , then we utilize Mironov’s key reconstruction algorithm, denoted by  $\text{GenKeys}$ . We discuss this more in a moment. After selecting the key masks,  $\mathcal{B}_2$  runs  $\mathcal{A}_2$  using  $K$  and the generated key masks which results in outputting another message  $M'$ . Finally, adversary  $\mathcal{B}_2$  determines if it “guessed” correctly (i.e., if there is a collision between the  $(l - \delta)^{\text{th}}$  compression function application for  $M$  and the  $(l' - \delta)^{\text{th}}$  compression function application for  $M'$ ). The subroutine  $\text{RetColl}$  simply calculates the inputs to the compression function that (potentially) collide.

The next lemma asserts the correctness of the key mask generation algorithm.

**Lemma 5.12** *Let  $P_1 \cdots P_{l-1} \in D^+$  and  $j \in [1..l - 1]$  and  $K \in \{0, 1\}^k$ . Then the process*

$$Y \stackrel{\$}{\leftarrow} \{0, 1\}^n; \{K_i\}_1^{\mu-1} \stackrel{\$}{\leftarrow} \text{GenKeys}(P_1 \cdots P_{l-1}, C_0, K, j, Y)$$

Adversary $\mathcal{B}_1$	Adversary $\mathcal{B}_2(K, S)$
Run $\mathcal{A}_1$	$(S_A, P_1 \cdots P_l, \delta, Y) \leftarrow S$
$\mathcal{A}_1$ outputs $(M, S_A)$	If $\delta \neq 0$ then
$P_1, \dots, P_l \stackrel{d}{\leftarrow} \text{padCS}_s(M)$	$\{K_i\}_1^{\mu-1} \stackrel{s}{\leftarrow} \text{GenKeys}(P_1 \cdots P_{l-1}, K, l - \delta, Y)$
$\delta \stackrel{s}{\leftarrow} [0..l-1]; Y \stackrel{s}{\leftarrow} \{0, 1\}^n$	$K_\mu \stackrel{s}{\leftarrow} \{0, 1\}^n$
$S \leftarrow (S_A, P_1 \cdots P_l, \delta, Y)$	Else
If $\delta = 0$ then Ret $(IV_2 \parallel Y \parallel P_l, S)$	$\{K_i\}_1^{\mu-1} \stackrel{s}{\leftarrow} (\{0, 1\}^n)^{\mu-1}$
Ret $(Y \parallel P_{l-\delta}, S)$	$K_\mu \leftarrow Y \oplus f^{\text{ESh}}((K, \{K_i\}_1^{\mu-1}), P_1 \cdots P_{l-1})$
<u>procedure GenKeys(<math>P_1 \cdots P_{l-1}, K, j, Y</math>)</u>	Run $\mathcal{A}_2((K, \{K_i\}_1^\mu), S_A)$
00 For $i = 1$ to $\mu - 1$ do $K_i \leftarrow \perp$	$\mathcal{A}_2$ outputs $M'; P'_1, \dots, P'_l \stackrel{d}{\leftarrow} \text{padCS}_s(M')$
01 While $j > 0$ do	If $l' < \delta$ then Ret $\perp$
02 $D \stackrel{s}{\leftarrow} \{0, 1\}^n; i \leftarrow j - 2^{\nu(j)}$	$(X, X') \leftarrow \text{RetColl}(K, \{K_i\}_1^\mu, P_1 \cdots P_l, P'_1 \cdots P'_l, \delta)$
03 For $z = i + 1$ to $j - 1$ do	If $X \neq X'$ and $f_K(X) = f_K(X')$ then Ret $X'$
04 If $K_{\nu(z)} = \perp$ then $K_{\nu(z)} \stackrel{s}{\leftarrow} \{0, 1\}^n$	Ret $\perp$
05 If $i = 0$ then $C_i \leftarrow IV_1$	<u>procedure RetColl(<math>K, \{K_i\}_1^\mu, P_1 \cdots P_l, P'_1 \cdots P'_l, \delta</math>)</u>
06 Else $C_i \leftarrow f_K(D \parallel P_{i-1})$	$Y_{l-\delta-1} \leftarrow f^{\text{ESh}}((K, \{K_i\}_1^{\mu-1}), P_1 \cdots P_{l-\delta-1})$
07 For $z = i + 1$ to $j - 1$ do	$Y'_{l'-\delta-1} \leftarrow f^{\text{ESh}}((K, \{K_i\}_1^{\mu-1}), P'_1 \cdots P'_{l'-\delta-1})$
08 $C_z \leftarrow f_K((C_{z-1} \oplus K_{\nu(z)}) \parallel X_z)$	If $\delta = 0$ then
09 $K_{\nu(j)} \leftarrow C_{j-1} \oplus Y$	$X \leftarrow (IV_2 \oplus K_0) \parallel (Y_{l-1} \oplus K_\mu) \parallel P_l$
10 $Y \leftarrow D; j \leftarrow i$	$X' \leftarrow (IV_2 \oplus K_0) \parallel (Y'_{l'-1} \oplus K_\mu) \parallel P'_l$
11 For $i = 0$ to $\mu - 1$ do	Else
12 If $K_i = \perp$ then $K_i \stackrel{s}{\leftarrow} \{0, 1\}^n$	$X \leftarrow (Y_{l-\delta-1} \oplus K_{\nu(l-\delta)}) \parallel P_{l-\delta}$
13 Ret $\{K_i\}_1^\mu$	$X' \leftarrow (Y'_{l'-\delta-1} \oplus K_{\nu(l'-\delta)}) \parallel P'_{l'-\delta}$
	Ret $(X, X')$

Figure 6: Adversary used in proof of Theorem 5.11.

has output distributed identically to  $\{K_i\}_1^{\mu-1} \stackrel{s}{\leftarrow} (\{0, 1\}^n)^\mu$ . Moreover, the key masks generated are such that  $Y = f^{\text{ESh}}((K, \{K_i\}_1^{\mu-1}), P_1 \cdots P_{j-1}) \oplus K_{\nu(j)}$ .

A proof of this lemma can be derived from the proof of Theorem 5 in [20]. For completeness we give a proof in Appendix A. Lemma 5.12 implies that  $\mathcal{B}_2$  simulates for  $\mathcal{A}_2$  exactly the environment it expects and that the target message  $\mathcal{B}_1$  committed to is consistent with  $\mathcal{A}_1$ 's target message  $M$  and keys  $K, \{K_i\}_1^\mu$ .

We now show that  $\mathcal{A}$  succeeding with probability  $\epsilon'$  means  $\mathcal{B}$  succeeds with probability  $\epsilon'/\tau(L)$  where  $\tau(L) \geq l$  is the number of blocks used for the maximal message length output by  $\mathcal{A}$ . This will imply the theorem statement. It suffices to argue that any pair of messages  $M \neq M'$  output by  $\mathcal{A}$  for which  $\text{ESh}^f((K, \{K_i\}_1^\mu), M) = \text{ESh}^f((K, \{K_i\}_1^\mu), M')$  implies a collision against the compression function. More specifically, letting  $P_1 \cdots P_l \stackrel{d}{\leftarrow} \text{padCS}_s(M)$  and  $P'_1 \cdots P'_l \stackrel{d}{\leftarrow} \text{padCS}_s(M')$  then there exists a  $\delta \in [0.. \min\{l, l'\} - 1]$  such that  $f_K(Y_{l-\delta-1} \parallel P_{l-\delta}) = f_K(Y'_{l'-\delta-1} \parallel P'_{l'-\delta})$  but  $Y_{l-\delta-1} \parallel P_{l-\delta} \neq Y'_{l'-\delta-1} \parallel P'_{l'-\delta}$ . Here the values  $Y_{l-\delta-1}$  and  $Y'_{l'-\delta-1}$  are as computed in RetColl. We consider several cases. If  $|M| \neq |M'|$  and  $d \geq n + 64$ , then necessarily  $P_l \neq P'_l$  due to strengthening. Thus  $\delta = 0$  specifies a collision against the compression function. If  $|M| \neq |M'|$  and  $d < n + 64$ , then  $\text{padCS}_s$  ensures that  $P_l = P'_l = 0^{d-n}$ . If  $Y_{l-1} \neq Y'_{l-1}$  then we are done with  $\delta = 0$ . Otherwise this implies that  $Y_{l-1} \oplus K_\mu = Y'_{l-1} \oplus K_\mu$  which means the output of the internal Shoup iteration is

equal for both messages. Since the strengthening in this case guarantees that  $P_{l-1} \neq P'_{l-1}$  we have that  $\delta = 1$  suffices.

If  $|M| = |M'|$ , then starting from the end of both messages, one can track backwards until finding the first value  $\delta$  for which the inputs to the compression function differ. We are guaranteed to find such a  $\delta$ , because otherwise it would imply that all message blocks for  $M$  and  $M'$  are equal, contradicting the fact that  $\mathcal{A}$  found a collision. Thus,  $\mathcal{B}$  has at least a  $1/l$  probability of guessing  $\delta$  correctly, and replacing  $l$  with  $\tau(L)$  gives that  $\mathcal{B}$  has advantage  $\epsilon'/\tau(L)$ .

Adversary  $\mathcal{B}$  runs  $\mathcal{A}$  and must additionally compute the compression function a number of times proportional to the number of blocks of the largest message output by  $\mathcal{A}$ . (Note that the while loop in `GenKeys` at line 01 never iterates more than  $l$  times, because  $i$  is always less than  $j$ .) ■

## References

- [1] J. An and M. Bellare. Constructing VIL-MACs from FIL-MACs: Message authentication under weakened assumptions. *Advances in Cryptology – CRYPTO 1999*, LNCS vol. 1666, Springer, pp. 252–269, 1999.
- [2] M. Bellare. New Proofs for NMAC and HMAC: Security Without Collision-Resistance. *Advances in Cryptology – CRYPTO 2006*, LNCS vol. 4117, Springer, pp. 113–120, 2006.
- [3] M. Bellare, R. Canetti, and H. Krawczyk. Keying Hash Functions for Message Authentication. *Advances in Cryptology – CRYPTO 1996*, LNCS vol. 1109, Springer, pp. 1–15, 1996.
- [4] M. Bellare, R. Canetti, and H. Krawczyk. Pseudorandom functions revisited: the cascade construction and its concrete security. *Proceedings of the 37th Annual Symposium on Foundations of Computer Science – FOCS 1996*, IEEE Computer Society, pp. 514–523, 1996.
- [5] M. Bellare and T. Ristenpart. Multi-Property-Preserving Hash Domain Extension and the EMD Transform. *Advances in Cryptology – ASIACRYPT '06*, LNCS vol. 4284, Springer, pp. 299–314, 2006.
- [6] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. *First ACM Conference on Computer and Communications Security – CCS '93*, ACM Press, pp. 62–73, 1993.
- [7] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption. *Advances in Cryptology – EUROCRYPT '94*, LNCS vol. 950, Springer, 92–111, 1994.
- [8] M. Bellare and P. Rogaway. The Exact Security of Digital Signatures - How to Sign with RSA and Rabin. *Advances in Cryptology – EUROCRYPT '96*, LNCS vol. 1070, Springer, pp. 399–416, 1996.
- [9] M. Bellare and P. Rogaway. Collision-Resistant Hashing: Towards Making UOWHF's Practical. *Advances in Cryptology – CRYPTO '97*, LNCS vol. 1294, Springer, pp. 470–484, 1997.
- [10] M. Bellare and P. Rogaway. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. *Advances in Cryptology – EUROCRYPT '06*, LNCS vol. 4004, Springer, pp. 409–426, 2006.
- [11] J.S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. *Advances in Cryptology – CRYPTO '05*, LNCS vol. 3621, Springer, pp. 21–39, 2004.
- [12] I. Damgård. A design principle for hash functions. *Advances in Cryptology – CRYPTO '89*, LNCS vol. 435, Springer, pp. 416–427, 1989.
- [13] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. Internet RFC 4346, 2006.

- [14] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). Internet RFC 2409, 1998.
- [15] S. Kent and K. Seo. Security Architecture for the Internet Protocol. Internet RFC 2401, 2005.
- [16] U. Maurer, R. Renner, and C. Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. *Theory of Cryptography – TCC '04*, LNCS vol. 2951, Springer, pp. 21–39, 2004.
- [17] U. Maurer and J. Sjödin. Domain Expansion of MACs: Alternative Uses of the FIL-MAC. *Cryptography and Coding 2005*, LNCS vol. 3796, Springer, pp. 168–185, 2005.
- [18] U. Maurer and J. Sjödin. Single-key AIL-MACs from any FIL-MAC. *International Colloquium on Automata, Languages, and Programming – ICALP '05*, LNCS vol. 3580, Springer, pp. 472–484, 2005.
- [19] R. Merkle. One way hash functions and DES. *Advances in Cryptology – CRYPTO '89*, LNCS vol. 435, Springer, pp. 428–446, 1989.
- [20] I. Mironov. Hash functions: from Merkle-Damgård to Shoup. *Advances in Cryptology – EUROCRYPT '01*, LNCS vol. 2045, Springer, pp. 166–181, 2001.
- [21] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. *Proceedings of the 21st Annual ACM Symposium on Theory of Computing – STOC'89*, ACM press, pp. 33–43, 1989.
- [22] National Institute of Standards and Technology. FIPS PUB 180-1: Secure Hash Standard. (1995) Supersedes FIPS PUB 180 1993 May 11.
- [23] RSA Laboratories. RSA PKCS #1 v2.1: RSA Cryptography Standards (2002).
- [24] R. Rivest. The MD4 Message Digest Algorithm. *Advances in Cryptology – CRYPTO'90*, LNCS vol. 537, Springer, pp. 303–311, 1990.
- [25] P. Rogaway. Formalizing Human Ignorance: Collision-Resistant Hashing without the Keys. *Vietcrypt '06*, LNCS vol. 4341, Springer, pp. 221–228, 2006.
- [26] P. Rogaway and T. Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. *Fast Software Encryption – FSE '04*, LNCS vol. 3017, Springer, pp. 371–388, 2004.
- [27] Sarkar, P.: Masking Based Domain Extenders for UOWHFs: Bounds and Constructions. In: Cryptology ePrint Archive, Report 2003/255 (2003), <http://eprint.iacr.org/>.
- [28] V. Shoup. A Composition Theorem for Universal One-Way Hash Functions. *Advances in Cryptology – EUROCRYPT '00*, LNCS vol. 1807, Springer, pp. 445–452, 2000.
- [29] G. Tsudik. Message Authentication with One-way Hash Functions. *SIGCOMM Comp. Commun. Rev.* 22, 5 (Oct. 1992), pp. 29–38.
- [30] X. Wang, Y. Yin, and H. Yu: Finding Collisions in the Full SHA-1. *Advances in Cryptology – CRYPTO '05*, LNCS vol. 3621, Springer, pp. 17–36, 2005.
- [31] X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. *Advances in Cryptology – EUROCRYPT '05*, LNCS vol. 3494, Springer, pp. 19–35, 2005.

## A Proof of Lemma 5.12

The proof of the lemma follows from results in [20], but for completeness we provide a detailed proof here. We first describe the algorithm (refer back to Figure 6) at a high level. It takes as input a target message, a compression function key, a target index, and a target chaining variable. The algorithm initially marks all the key masks as undefined (line 00). The main loop begins by choosing a new target chaining variable  $D$  on line 02. It then calculates an index  $i$ . This is the least index less than  $j$  for which there does not exist a  $c \in [i + 1 .. j - 1]$  such that  $\nu(c) = \nu(i)$  or



$\nu(c) = \nu(j - 1)$ . This ensures that the algorithm is free to set the masks used between blocks  $i$  and  $j - 1$  to be uniformly chosen values, which the algorithm does next (lines 03 and 04). Then it computes the value  $C_{j-1}$  (lines 07–08) using the just specified masks, the target message, and an initial  $C_i$  value that is either  $IV_1$  (line 05) or the output of the compression function applied to the new target chaining variable and the appropriate message block (line 06). It uses  $C_{j-1}$  and the target chaining variable  $Y$  to set mask  $K_{\nu(j-1)}$  (line 09). At this point the algorithm has ensured consistency starting from location  $i$ , but if  $i > 0$  then it now must recurse and ensure consistency with the new target chaining variable.

We justify the first part of Lemma 5.12: the output distribution of  $\text{GenKeys}(P_1 \cdots P_{t-1}, K, j, Y)$  for  $Y \xleftarrow{\$} \{0, 1\}^n$  is identical to choosing  $K_i$  uniformly for each  $i \in [1..t]$ . Values assigned to variables  $K_i$  due to lines 04 and 12 clearly sample uniformly from  $\{0, 1\}^n$ . Assignments to variables  $K_i$  due to line 08 inherit the distribution of  $Y$ , which is always distributed uniformly.

To prove the second part of the lemma, we note that as long as no key mask is ever defined more than once, then lines 05–09 of the algorithm ensure that  $Y = f^{\text{ESh}}((K, \{K_i\}_1^{\mu-1}), P_1 \cdots P_{j-1}) \oplus K_{\nu(j)}$ . Thus all that remains is to show that  $\text{GenKeys}$  never redefines key masks. We first establish two invariants of the algorithm.

**Invariant 1:**  $\nu(i) > \nu(j)$

We have that  $j = (2b + 1)2^{\nu(j)}$  for a number  $b \geq 0$  (note that the coefficient of  $2^{\nu(j)}$  cannot be even, since otherwise  $\nu(j)$  would not be the maximum power of 2 dividing  $j$ ). Then  $i = j - 2^{\nu(j)} = (2b + 1)2^{\nu(j)} - 2^{\nu(j)} = (2b)2^{\nu(j)} = b2^{\nu(j)+1}$ . This implies that  $\nu(i) > \nu(j)$ .

**Invariant 2:**  $\nu(j) > \nu(l)$  for  $i < l < j$

We have that  $i = (2b)2^{\nu(j)}$  (as shown above) and that  $j = (2b + 1)2^{\nu(j)}$ . Assume for contradiction that there exists an  $l$  with  $\nu(l) \geq \nu(j)$  and  $i < l < j$ . Then  $l = k2^{\nu(j)+d} = 2^d k 2^{\nu(j)}$  for some  $k \geq 1$  and  $d \geq 0$ . But by our restriction on  $l$  and substituting in the just established equivalences for  $i$ ,  $l$ , and  $j$  we have that

$$(2b)2^{\nu(j)} < 2^d k 2^{\nu(j)} < (2b + 1)2^{\nu(j)} \Rightarrow 2b < 2^d k < 2b + 1$$

but this is not possible, and so we have a contradiction.

Now we argue that masks are never redefined. Key mask assignments on lines 04 or 11 are guarded by if statements and thus cannot redefine a key mask. Invariant 1 implies that  $\nu(j)$  is always increasing, and invariant 2 implies that  $\nu(j)$  is always larger than any key mask assigned in line 04. Together this implies that line 09 will never redefine a key mask, completing the proof.