Multi-Property-Preserving Hash Domain Extension: The EMD Transform

Mihir Bellare and Thomas Ristenpart*

August 4, 2006

Abstract

In this paper we (1) argue the benefits of replacing the current MD transform with a *multi-property-preserving domain extension transform* that guarantees numerous properties of the hash function assuming they hold of the compression function; (2) provide a practical, proven-secure multi-domain extension transform suitable for use with the next generation of hash functions; (3) point to some subtle weaknesses in the transforms of Coron et al. [9] that imply they are not only not suitable multi-property transforms but in fact in some ways provide lower security guarantees than even the current MD transform.

1 Introduction

Background. Recall that hash functions are built in two steps. First, one designs a compression function h: $\{0,1\}^{d+n} \to \{0,1\}^n$, where d is the length of a data block and n is the length of the chaining variable. Then one specifies a *domain extension transform* H that utilizes h as a black box to implement the hash function H^h: $\{0,1\}^* \to \{0,1\}^n$ associated to h. All current hash functions use the Merkle-Damgård (MD) transform [12, 10] because it has been proven [12, 10] to be *collision-resistance preserving* (CR-Pr): if h is collision-resistant (CR) then so is H^h. This means that the cryptanalytic validation task can be confined to the compression function.

A rising bar. Current usage makes it obvious that CR no longer suffices as the security goal for hash functions. In order to obtain MACs and PRFs, hash functions were keyed. 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.) Hash functions are also used to instantiate random oracles [4] in public-key schemes such as RSA-OAEP [5] and RSA-PSS [6] in the RSA PKCS#1 v2.1 standard [16]. CR is insufficient for arguing the security of hash function based MACs or PRFs, let alone hash-function based random oracles. And it does not end there. Whether hash function designers like it or not, application builders will use hash functions for all kinds of tasks that presume beyond-CR properties. Not all such uses can be sanctified, but the central and common ones should be. We think that the type of usage we are seeing for hash functions will continue, and it is in the best interests of security to make the new hash functions rise as far towards this bar as possible, by making them strong and versatile tools that have security attributes beyond CR.

How to get there. The two-step design paradigm in current use is compelling because it reduces the cryptanalytic task of providing CR of the hash function to certifying only that the compression function has the same property. It makes sense to seek other attributes via the appropriate extension of this paradigm. To spell it out, if we want a hash function with properties $P_1, \ldots, P_n$ then we should (1) design a compression function h with the goal of having properties $P_1, \ldots, P_n$, and (2) apply a domain extension transform H that *provably* preserves P_i for every $i \in [1..n]$. We call such a compression function a multi-property one, and we call such a transform a *multi-property-preserving domain extension transform.* Note that we want a *single* transform that preserves multiple properties, resulting in a single, multi-property hash function, as opposed to a transform per property which would result in not one but numerous hash functions.

This work. The goal of our paper is to point to the need for a multi-property domain extension transform, point to the properties it would be desirable for such a transform to preserve, point to some weaknesses of existing transforms in this regard, and finally propose a simple, new multi-property preserving domain extension transform that is suitable for standards.

Whence the compression function? We do not address the problem of constructing a multi-property compression function. We presume that this can and will be done, and focus on the domain extension problem. Our confidence in the emergence of strong compression functions might be questioned in the light of the recent collision-finding attacks [18, 17] that have destroyed some hash functions and tainted others. But we do not feel a need for pessimism. We recall the story for block ciphers, where the AES yielded by the NIST competition was not only faster than DES but seems stronger and more elegant. We believe it will be the same for compression functions. Namely, we believe that the cryptanalytic talent in

*Department of Computer Science and Engineering, UC San Diego, {mihir,tristenp}@cs.ucsd.edu

our community will yield compression functions having the properties (CR and beyond) that we want, and perhaps without increase, or even with decrease, in cost, compared to current hash functions, just as happened for block ciphers. Rather than being conservative, we feel one should take advantage of the chance to update our hash functions by raising the bar in terms of requirements. We wish to contribute to this process on the domain extension transform side.

PROPERTIES TO PRESERVE. The first question to ask is which properties our multi-property domain extension transform should preserve. We wish, of course, that the transform continue to be CR-Pr, meaning that it preserve CR. Another obvious desirable property is that it be pseudorandom function preserving (PRF-Pr). That is, if an appropriately keyed version of the compression function is a PRF then the appropriately keyed version of the hash function must be a PRF too. This last goal is important due to the many uses of hash functions as MACs and PRFs via keying as mentioned above. The final goal we will ask is that the transform be pseudorandom-oracle preserving (PRO-Pr) [9].

WEAKNESSES OF THE PRO-Pr TRANSFORMS OF [9]. Before we can explain what we provide we need to explain some things about PRO-Pr, what is provided by the transforms of [9], and, more importantly, what is *not* provided. Specifically, we will argue that the PRO-Pr transforms of [9] are unsuitable for current use because they fail to be CR-Pr. This point is somewhat subtle so needs some explanation.

We use the moniker *pseudorandom oracle* for any construction that is indifferentiable from a random oracle as defined in [11]. A transform is PRO-Pr if it preserves the property of "behaving like a random oracle": if h is modeled as a random oracle, then H^h should be a pseudorandom oracle. As explained at length in [9], this is a desirable property to support the usages of hash functions as random oracles. We agree.

PRO-Pr seems like a very strong property to have, and at first glance one is tempted to think that it is the only property a transform need preserve because it automatically guarantees that the constructed hash function has many nice properties. For example, the hash function would be CR. It could be keyed in almost any way to yield a PRF and MAC. And so on. This would be true, because random oracles have these properties, and hence so do pseudorandom oracles. However, we point out that the above reasoning is flawed and there is a danger to PRO-Pr in practice. Namely, the fact that a transform is PRO-Pr *does not guarantee* that the constructed hash function is CR, even if the compression function is CR. We demonstrate this with a counter-example. Namely we give an example of a transform that is PRO-Pr, yet there is a CR compression function such that the hash function resulting from the transform is not CR. That is, the transform is PRO-Pr but not CR-Pr, or, in other words, PRO-Pr does not imply CR-Pr.

What this shows is that using a PRO-Pr transform could be *worse* than using the standard, strengthened Merkle-Damgård transform from the point of view of security because at least the latter guarantees the hash function is CR if the compression function is, but the former does not. If we blindly move to PRO-Pr transforms, our security guarantees are actually going down, not up.

How can this be? It comes about because PRO-Pr provides guarantees only if the compression function is a random oracle or pseudorandom oracle. But of course any real compression function is *provably not* either of these. (One can easily differentiate it from a random oracle because it can be computed by a small program.) Thus, when a PRO-Pr transform works on a real compression function, we have essentially no provable guarantees on the resulting hash function. This is in some ways analogous to the kinds of issues pointed out in [8, 13] about the sometimes impossibility of instantiating random oracles.

The fact that a PRO-Pr transform need not in general be CR-Pr does not mean that some *particular* PRO-Pr transform is not CR-Pr. We therefore investigate each of the four PRO-Pr schemes suggested by Coron et al. The schemes make slight modifications to the MD transform: the first applies a prefix-free encoding, the second 'throws' away some of the output, and the third and fourth utilize an extra compression function application. These modifications serve to render the MD transform PRO-Pr. Unfortunately, we show that none of the four transforms is CR-Pr. In each case, we present an example CR compression function such that the hash function yielded by the transform is *not* CR. In particular, this means that these transforms do not provide the same guarantee as the existing and in-use Merkle-Damgård transform. For this reason we think these transforms should not be considered suitable for use in the design of new hash functions.

CLARIFICATION. We clarify that we are *not* suggesting that the pseudorandom oracle preservation goal of [9] is unimportant or should not be achieved. In fact we think it is a very good idea and should be a property of any new transform. This is so because in cases where we are (heuristically) assuming the hash function is a random oracle, this goal reduces the assumption to the compression function being a random oracle. What we have shown above is that, *by itself*, it is not enough because it weakens existing, standard-model guarantees. This is why our goal is a multi-property domain extension transform, in particular one that is not only PRO-Pr, but also CR-Pr and PRF-Pr. To be clear, we ask that, for a transform H to be considered suitable, one should do the following. First, prove

that H^h is CR using only the fact that h is CR. Then show that H^h is a pseudorandom oracle when h is a pseudorandom oracle. Finally, use some natural keying strategy to key H^h and assume that h is a good PRF, then prove that H^h is also a good PRF.

New transform. There is to date no transform with all the properties above. (Namely, that it is PRO-Pr, CR-Pr and PRF-Pr.) The next contribution of this paper is a new transform EMD (Enveloped Merkle-Damgård) which is the first to meet our definition of hash domain extension security: EMD is proven to be CR-Pr, PRO-Pr, and PRF-Pr. The transform is simple and easy to implement in practice (see the figure in Section 5). It is based on the MD transform and combines two mechanisms to ensure that it preserves all the properties of interest. The first mechanism is the (well-known) Merkle-Damgård strengthening: we always concatenate an input message with the 64-bit encoding of its length. This ensures that EMD is CR-Pr. The second mechanism is the use of an 'envelope' to hide the internal MD iteration. This technique has been proposed previously (e.g., it is used in NMAC and HMAC [3]). In general, the technique involves applying a distinct compression function (or the same compression function, but in some distinguished way) to the output of an MD iteration. Our enveloping technique is novel in two ways. First, we allow adversarially-controlled message bits to be input into the envelope, which increases the efficiency of the scheme. Second, we utilize a novel reduction technique in our proof that EMD is PRO-Pr to show that simply fixing n bits of the envelope's input is sufficient to cause the last application of the random oracle to behave independently with high probability. This solution allows our transform to work on a single compression function without requiring the costly work-arounds previously suggested (e.g., prefix-free encodings or prepending a block of zeros to input messages).

Patching existing transforms. We remark that it is possible to patch the transforms of [9] so that they are CR-Pr. Namely, one could use Merke-Damgård strengthening, which they omitted. However our transform still has several advantages over their transforms. One is that ours is cheaper, i.e. more efficient, and this matters in practice. Another is that ours is PRF-Pr. A result of [1] implies that one of the transforms of [9] is PRF-Pr, but whether or not this is true for the others is not clear.

Summary. Figure 1 summarizes our quantitative results. We now recap our main contributions. First, we show that property preservation by domain extension transforms is orthogonal: a transform that preserves one security property does not necessarily preserve others. In particular, we show that pseudorandom oracle preservation does not imply preservation of collision resistance or PRF preservation. We reinforce this general result by showing that the PRO-Pr transforms proposed by Coron et al. are not CR-Pr. We propose that a domain extension transform only be considered secure if it is simultaneously collision-resistance preserving, pseudorandom oracle preserving, *and* pseudorandom function preserving. Finally we propose a simple and efficient transform EMD that is the first to meet all three conditions of security. We utilize the technique of 'enveloping' in novel ways to increase the efficiency and simplicity of EMD as compared to previously proposed PRO-Pr schemes.

2 Definitions

Notation. Let $D = \{0,1\}^d$ and $D^+ = \cup_{i\geq 1}\{0,1\}^{id}$. We denote pairwise concatenation by $\|$, e.g. $M \| M'$. We will often write the concatenation of a sequence of string by $M_1 \cdots M_k$, which translates to $M_1 \| M_2 \| \ldots \| M_k$. For brevity, we define the following semantics for the notation $M_1 \cdots M_k \xleftarrow{d} M$ where M is a string of $|M|$ bits: 1) define $k = \lceil |M|/d \rceil$ and 2) if $|M| \bmod d = 0$ then parse M into $M_1, M_2, \ldots, M_k$ where $|M_i| = d$ for $1 \leq i \leq k$, otherwise parse M into $M_1, M_2, \ldots, M_{k-1}, M_k$ where $|M_i| = d$ for $1 \leq i \leq k-1$ and $|M_k| = |M| \bmod d$. For any finite set S we write $s \xleftarrow{\$} S$ to signify uniformly choosing a value $s \in S$.

Oracle TMs, Random Oracles, and transforms. Cryptographic schemes, adversaries, and simulators are modeled as Oracle Turing Machines (OTM) and are possibly given zero or more oracles, each being either a random oracle or another OTM (note that when used as an oracle, an OTM maintains state across queries). We allow OTMs to expose a finite number of interfaces: an OTM $\mathsf{N} = (\mathsf{N}_1, \mathsf{N}_2, \ldots, \mathsf{N}_l)$ exposes interfaces $\mathsf{N}_1, \mathsf{N}_2, \ldots, \mathsf{N}_l$. For brevity, we write M^{N} to signify that M gets to query all the interfaces of N.

For a set *Dom* and finite set *Rng* we define a *random function* by the following TM accepting inputs $X \in Dom$:

Algorithm $\mathsf{RF}_{Dom,Rng}(X)$:
  if $T[X] = \perp$ **then** $T[X] \xleftarrow{\$} Rng$
  return $T[X]$

where T is a table everywhere initialized to $\perp$. This implements a random function via lazy sampling (which allows us to reason about the case in which *Dom* is infinite). In the case that $Dom = \{0,1\}^d$ and $Rng = \{0,1\}^r$ we write $\mathsf{RF}_{d,r}$ in place of $\mathsf{RF}_{Dom,Rng}$. We similarly define $\mathsf{RF}_{d,Rng}$ and $\mathsf{RF}_{Dom,r}$ in the obvious ways and write $\mathsf{RF}_{*,r}$ in the special case that $Dom = \{0,1\}^*$. A *random oracle* is simply a public random function: all parties (including the adversary) are given access. We write $f, g, \ldots = \mathsf{RF}_{Dom,Rng}$ to signify that f, g, ... are independent random oracles from *Dom* to *Rng*.

Transform	CR-Pr	PRO-Pr	PRF-Pr	Number of calls to h to hash M, $\lvert M\rvert = b \geq d$
Plain MD (MD)	No	No	No	$N(b) = \left\lceil \frac{b}{d} \right\rceil$
Strengthened MD (SMD)	[12, 10]	No	No	$N(b) = \begin{cases} \left\lceil \frac{b}{d} \right\rceil & \text{if } b \bmod d < d-64 \\ \left\lceil \frac{b}{d} \right\rceil + 1 & \text{otherwise} \end{cases}$
Prefix-Free (PRE)	No	[9]	[1]	$N(b) = \left\lceil \frac{b}{d-1} \right\rceil$
Chop Solution (CHP)	No	[9]	?	$N(b) = \left\lceil \frac{b}{d} \right\rceil$
NMAC Construction (NT)	No	[9]	?	$N(b) = \left\lceil \frac{b}{d} \right\rceil + 1$
HMAC Construction (HT)	No	[9]	?	$N(b) = \left\lceil \frac{b}{d} \right\rceil + 2$
Enveloped MD (EMD)	[12]	Thm. 2	Thm. 3	$N(b) = \begin{cases} \left\lceil \frac{b}{d} \right\rceil & \text{if } b \bmod d < d-n-64 \\ \left\lceil \frac{b}{d} \right\rceil + 1 & \text{otherwise} \end{cases}$

Figure 1: Comparison of transform security and efficiency when applied to h: $\{0,1\}^{n+d} \to \{0,1\}^n$. The last column specifies the number of calls to h needed to hash a b-bit message M under each transform (where $b \geq d$).

A *transform* C describes how to utilize an arbitrary compression function to create a variable-input-length hash function. When we fix a particular compression function f, we get the associated cryptographic scheme $C^f \equiv C[f]$.

Collision resistance. We consider a function F to be collision resistant if it is computationally infeasible to find any two messages $M \neq M'$ such that $F(M) = F(M')$. For the rest of the paper we use h to always represent a collision-resistant hash function with signature h: $\{0,1\}^{d+n} \to \{0,1\}^n$.

PRFs. Let F: $Keys \times Dom \to Rng$ be a function family. Informally, we consider F a pseudorandom function family (PRF) if no reasonable adversary can succeed with high probability at distinguishing between $F(K,\cdot)$ for $K \xleftarrow{\$} Keys$ and a random function $f = \mathsf{RF}_{Dom,Rng}$. More compactly we write the *prf-advantage* of an adversary A as

$$\mathbf{Adv}^{\mathrm{prf}}_{F}(A) = \Pr\left[K \xleftarrow{\$} Keys; A^{F(K,\cdot)} \Rightarrow 1\right] - \Pr\left[A^{f(\cdot)} \Rightarrow 1\right]$$

where the probability is taken over the random choice of K and the coins used by A or by the coins used by f and A. For the rest of the paper we use e to always represent a PRF with signature e: $\{0,1\}^{d+n} \to \{0,1\}^n$ that is keyed through the low n bits of the input.

PROs. The indifferentiability framework [11] generalizes the more typical indistinguishability framework (e.g., our definition of a PRF above). The new framework captures the necessary definitions for comparing an object that utilizes public components (e.g., fixed-input-length (FIL) random oracles) with an ideal object (e.g., a variable-input-length (VIL) random oracle). Fix some number l. Let $C^{f_1,\ldots,f_l}$: $Dom \to Rng$ be a function for random oracles $f_1, \ldots, f_l = \mathsf{RF}_{D,R}$. Then let $S^{\mathcal{F}} = (S_1, \ldots, S_l)$ be a simulator OTM with access to a random oracle $\mathcal{F} = \mathsf{RF}_{Dom,Rng}$ and which exposes interfaces for each random oracle utilized by C. (The simulator's goal is to mimic $f_1, \ldots, f_l$ in such a way as to convince an adversary that $\mathcal{F}$ is C.) The *pro-advantage* of an adversary A against C is the difference between the probability that A outputs a one when given oracle access to $C^{f_1,\ldots,f_l}$ and $f_1, \ldots, f_l$ and the probability that A outputs a one when given oracle access to $\mathcal{F}$ and $S^{\mathcal{F}}$. More succinctly we write that the pro-advantage of A is

$$\mathbf{Adv}^{\mathrm{pro}}_{C,S}(A) = \left|\Pr\left[A^{C^{f_1,\ldots,f_l},f_1,\ldots,f_l} \Rightarrow 1\right] - \Pr\left[A^{\mathcal{F},S^{\mathcal{F}}} \Rightarrow 1\right]\right|$$

where, in the first case, the probability is taken over the coins used by the random oracles and A and, in the second case, the probability is over the coins used by the random oracles, A, and S. For the rest of the paper we use f to represent a random oracle $\mathsf{RF}_{d+n,n}$.

Resources. We do not give formal notions of security (e.g., in terms of negligible functions) and rather give concrete statements about the advantage of adversaries using certain resources. For prf-adversaries we measure the total number of queries q made and the running time t. For pro-adversaries we measure the total number of *left queries* q_L (which are either to C or $\mathcal{F}$) and the number of *right queries* q_i made to each oracle f_i or simulator interface S_i. We also specify the resources utilized by simulators. We measure the total number of queries q_S to $\mathcal{F}$ and the maximum running time t_S. Note that these values are generally functions of the number of queries made by an adversary (necessarily so, in the case of t_S).

Pointless queries. In all of our proofs (for all notions of security) we assume that adversaries make no *pointless queries*. In our setting this particularly means that adver-

Algorithm $c^+(I, M)$:
  $M_1 \cdots M_k \xleftarrow{d} M$; $Y_0 \leftarrow I$
  for $i = 1$ to k **do**
    $Y_i \leftarrow c(M_i \,\|\, Y_{i-1})$
  return Y_k

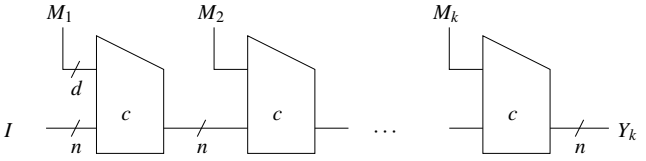


Figure 2: The Merkle-Damgård iteration.

saries are never allowed to repeat a query to an oracle.

3 Domain Extension using Merkle-Damgård

The Merkle-Damgård transform. We focus on variants of the Merkle-Damgård transform. Using an arbitrary fixed-input-length function c: $\{0,1\}^{d+n} \to \{0,1\}^n$ we wish to construct a family of variable-input-length functions F^c: $\{0,1\}^n \times \{0,1\}^* \to \{0,1\}^n$. We start by defining the Merkle-Damgård iteration c^+: $D^+ \to \{0,1\}^n$ by the algorithm specified in Figure 2.

The function c^+ only works on strings that are a multiple of d bits, yet we seek schemes that will work on arbitrary (or close to arbitrary) string lengths. Thus we require a padding function $\mathsf{pad}(M)$, which for any string $M \in \{0,1\}^*$ returns a string Y of length $d\lceil |M|/d \rceil$. We assume that pad is one-to-one (this assumption is made for all padding functions in this paper). Fixing some $IV \in \{0,1\}^n$, we define the *plain Merkle-Damgård transform* $\mathrm{MD}[c] = c^+(IV, \mathsf{pad}(\cdot))$.

Keying strategies. In this paper we discuss transforms that produce keyless schemes. We would also like to utilize these schemes as variable-input-length PRFs, but this requires that we use some keying strategy. We focus on the *key-via-IV strategy*. Under this strategy, we replace constant initialization vectors with randomly chosen keys of the same size. For example, if e is a particular PRF, then keyed MD^e would be defined as $\mathrm{MD}^e_K(M) = e^+(K, \mathsf{pad}(M))$. (it should be noted that this is not a secure PRF). We will always signify the keyed version of a construction by explicitly including the keys as subscripts.

Multi-property preservation. We would like to reason about the security of MD and its variants when we make assumptions about c. Phrased another way, we want to know if a transform such as MD *preserves* security properties of the underlying compression function. We are interested in collision-resistance preservation, PRO preservation, and PRF preservation. Let C be a transform that works on functions from $\{0,1\}^{d+n}$ to $\{0,1\}^n$. Let h: $\{0,1\}^{d+n} \to \{0,1\}^n$ be a collision-resistant hash function. Then we say that C is *collision-resistance preserving* (CR-Pr) if the scheme C^h is collision-resistant. Let $f = \mathsf{RF}_{d+n,n}$ be a random oracle. Then we say that C is *pseudorandom oracle preserving* (PRO-Pr) if the scheme C^f is a pseudorandom oracle. Let e: $\{0,1\}^{d+n} \to \{0,1\}^n$ be an arbitrary PRF (keyed via the low n bits). Then we say that C is *pseudorandom function preserving* (PRF-Pr) if the keyed-via-IV scheme C^e_K is a PRF.

Security of MD and SMD. It is well known that MD is neither CR-Pr, PRO-Pr, or PRF-Pr [12, 10, 1, 9]. The first variant that was proven CR-Pr was so-called MD with strengthening, which we denote by SMD. In this variant, the padding function is replaced by one with the following property: for M and M' with $|M| \neq |M'|$ then $M_k \neq M'_k$ (the last blocks after padding are distinct). A straightforward way to achieve a padding function with this property is to include an encoding of the message length in the padding. In many implementations, this encoding is done using 64 bits [15], which restricts the domain to strings of length no larger than 2^{64}. We therefore fix some padding function $\mathsf{pad64}(M)$ that takes as input a string M and returns a string Y of length $k\lceil |M|/d \rceil$ bits for some number k such that the last 64 bits of Y are an encoding of $|M|$. Using this padding function we define the *strengthened MD transform* $\mathrm{SMD}[c] = c^+(IV, \mathsf{pad64}(\cdot))$. We emphasize the fact that preservation of collision-resistance is *strongly dependent* on the choice of padding function. However, this modification to MD is alone insufficient for rendering SMD either PRF-Pr or PRO-Pr due to simple length-extension attacks [1, 9].

4 Orthogonality of Property Preservation

In this section we illustrate that property preservation is orthogonal. Previous work [9] has already shown that collision-resistance preservation does not imply pseudorandom oracle preservation. We investigate the inverse: does a transform being PRO-Pr imply that it is also CR-Pr? We answer this in the negative by showing how

to construct a PRO-Pr transform that is not CR-Pr. While this result is sufficient to refute the idea that PRO-Pr is a stronger security goal for transforms, it does not necessarily imply anything about actual PRO-Pr transforms. Thus, we investigate the four proposed by Coron et al. and show that all four fail to preserve collision-resistance. Finally, lacking a formally meaningful way of comparing pseudorandom oracle preservation and pseudorandom function preservation (one resulting in keyless schemes, the other in keyed), we briefly discuss whether the proposed transforms are PRF-Pr.

4.1 PRO-Pr does not imply CR-Pr

Let $n, d > 0$ and h: $\{0,1\}^{d+n} \to \{0,1\}^n$ be a collision-resistant hash function and $f = \mathsf{RF}_{d+n,n}$ be a random oracle. Let Dom, Rng be non-empty sets and let C_1 be a transform for which $C_1^f \equiv C_1[f]$ is a pseudorandom oracle C_1^f: $Dom \to Rng$. We create a transform C_2 that is PRO-Pr but is *not* CR-Pr. In other words the resulting scheme C_2^f: $Dom \to Rng$ is indifferentiable from a random oracle, but it is trivial to find collisions against the scheme C_2^h (even without finding collisions against h). We modify $C_1[c]$ to create $C_2[c]$ as follows. First check if $c(0^{d+n})$ is equal 0^n and return 0^n if that is the case. Otherwise we just follow the steps specified by $C_1[c]$. Thus the scheme C_2^f returns 0^n for any message if $f(0^{d+n}) = 0^n$. Similarly the scheme C_2^h returns 0^n for any message if $h(0^{d+n}) = 0^n$. The key insight, of course, is that the differing assumptions made about the oracle impact the likelihood of this occurring. If the oracle is a random oracle, then the probability is small: we prove below that C_2^f is a pseudorandom oracle. First we show how to easily design a collision-resistant hash function h that causes C_2^h to not be collision resistant. Let h': $\{0,1\}^{d+n} \to \{0,1\}^{n-1}$ be some collision-resistant hash function. Then $h(M)$ returns 0^n if $M = 0^{d+n}$, otherwise it returns $h'(M) \,\|\, 1$. Figure 3 summarizes the construction. Collisions found on h would necessarily translate into collisions for h', which implies that h is collision-resistant. Furthermore since $h(0^{d+n}) = 0^n$ we have that $C_2^h(M) = 0^n$ for any message M, making it trivial to find collisions against C_2^h.

Proposition 1 [C_2 is PRO-Pr] *Let $n, d > 0$ and Dom, Rng be non-empty sets and $f = \mathsf{RF}_{d+n,n}$ and $\mathcal{F} = \mathsf{RF}_{Dom,Rng}$ be random oracles. Let C_1^f be a pseudorandom oracle. Let C_2^f be the scheme as described above and let S be an arbitrary simulator. Then for any adversary A_2 that utilizes q_L left queries, q_R right queries, and runs in time t, there exists an adversary A_1 such that*

$$\mathbf{Adv}^{\mathrm{pro}}_{C_2,S}(A_2) \le \mathbf{Adv}^{\mathrm{pro}}_{C_1,S}(A_1) + \frac{1}{2^n}\,.$$

with A_1 utilizing the same number of queries and time as A_2.

Proof: Let $f = \mathsf{RF}_{d+n,n}$ and $\mathcal{F} = \mathsf{RF}_{Dom,Rng}$ be random oracles. Let A be some pro-adversary against C_2^f. Let S be an OTM with an interface S_f that on $(d+n)$-bit inputs returns n-bit strings. We utilize a simple game-playing argument in conjunction with a hybrid argument to bound the indifferentiability of C_2 by that of C_1 (with respect to simulator S). Figure 3 displays two games, game G0 (includes boxed statement) and game G1 (boxed statement removed). The first game G0 exactly simulates the oracles C_2^f and f. The second game G1 exactly simulates the oracles C_1^f and f. We thus have that $\Pr\ A^{C_2^f,f} \Rightarrow 1\ = \Pr\ A^{\mathrm{G0}} \Rightarrow 1\ $ and $\Pr\ A^{C_1^f,f} \Rightarrow 1\ = \Pr\ A^{\mathrm{G1}} \Rightarrow 1\ $. Since G0 and G1 are identical-until-bad we have by the fundamental lemma of game playing [7] that $\Pr\ A^{\mathrm{G0}} \Rightarrow 1\ - \Pr\ A^{\mathrm{G1}} \Rightarrow 1\ \le \Pr\ A^{\mathrm{G1}}$ sets bad . The right hand side is at most 2^{-n} because f is a random oracle. Thus,

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{pro}}_{C_2,S}(A_2) &= \Pr\ A^{\mathrm{G0}} \Rightarrow 1\ - \Pr\ A^{\mathcal{F},S^{\mathcal{F}}} \Rightarrow 1 \\
&= \Pr\ A^{\mathrm{G0}} \Rightarrow 1\ - \Pr\ A^{\mathrm{G1}} \Rightarrow 1\ + \\
&\quad \Pr\ A^{C_1^f,f} \Rightarrow 1\ - \Pr\ A^{\mathcal{F},S^{\mathcal{F}}} \Rightarrow 1 \\
&= \Pr\ A^{\mathrm{G1}} \text{ sets } bad\ + \Pr\ A^{C_1^f,f} \Rightarrow 1\ - \\
&\quad \Pr\ A^{\mathcal{F},S^{\mathcal{F}}} \Rightarrow 1 \\
&\le \frac{1}{2^n} + \mathbf{Adv}^{\mathrm{pro}}_{C_1,S}(A_1)
\end{aligned}$$

∎

4.2 Insecurity of Proposed PRO-Pr Transforms

Collision-resistance preservation. The result above tells us that PRO-Pr does not imply CR-Pr for arbitrary schemes. What about MD variants? One might hope that the mechanisms used to create an PRO-Pr MD variant are sufficient for rendering the variant CR-Pr also. This is not true. In fact *all* previously proposed MD variants proven to be PRO-Pr *are not* CR-Pr. The four variants are summarized in Figure 4, see [9] for more details.

The first transform is *Prefix-free MD* specified by $\mathrm{PRE}[c] = c^+(IV, \mathsf{padPF}(\cdot))$. It applies a prefix-free padding function padPF to an input message and then uses the MD iteration. The padding function padPF must output strings that are a multiple of d bits with the property that for any two strings $M \neq M'$, $\mathsf{padPF}(M)$ is not a prefix of $\mathsf{padPF}(M')$. The *Chop solution* simply drops s bits from the output of the MD iteration applied to a message. The *NMAC transform* applies a second, distinct compression function to the output of an MD iteration; it is defined by $\mathrm{NT}[c, g] = g(c^+(IV, \cdot))$, where g is a function from n bits to n bits (distinct from h). Lastly, the *HMAC Transform* is defined by $\mathrm{HT}[c] =$

Let h': $\{0,1\}^{n+d} \to \{0,1\}^{n-1}$ be CR. Then define h: $\{0,1\}^{n+d} \to \{0,1\}^n$ by

$$h(x) = \begin{cases} 0^n & \text{if } M = 0^{d+n} \\ h'(M) \,\|\, 1 & \text{otherwise} \end{cases}$$

procedure Initialize	**procedure** $C(X)$	Game G0	Game G1
000 $f = \mathsf{RF}_{d+n,n}$	200 $Y \leftarrow C_1^f(X)$		
procedure $f(x)$	201 **if** $f(0^{d+n}) = 0^n$ **then** $bad \leftarrow$ true;	$Y \leftarrow 0^n$	
100 **return** $f(x)$	202 **return** Y		

Figure 3: (Left) Definition of the collision-resistant compression function used in the proof of Proposition 1 and the counter-examples of Section 4.2. (Right) Games utilized in the proof of Proposition 1 to show that C_2^f is a pseudorandom oracle.

Prefix-free MD: $\mathrm{PRE}[c] = c^+(IV, \mathsf{padPF}(\cdot))$ where padPF: $\{0,1\}^* \to D^+$ is a prefix-free padding function	NMAC Transform: $\mathrm{NT}[c,g] = g(c^+(IV,\cdot))$ where g: $\{0,1\}^n \to \{0,1\}^n$ is a function
Chop Solution: $\mathrm{CHP}[c]$ = first $n-s$ bits of $c^+(IV, \mathsf{pad}(\cdot))$	HMAC Transform: $\mathrm{HT}[c] = c(c^+(IV, 0^d \,\|\|\, \cdot) \,\|\| \, 0^{d-n} \,\|\| \, IV)$

Figure 4: The four MD variants proposed in [9] that are PRO-Pr but not CR-Pr.

$c(c^+(IV, 0^d \,\|\| \cdot) \,\|\| \, 0^{d-n} \,\|\| \, IV)$. This transform similarly utilizes enveloping: the MD iteration is fed into c in a way that distinguishes this last call from the uses of c inside the MD iteration. The prepending of a d-bit string of zeros to an input message helps ensure that the envelope acts differently than the first compression function application.

To simplify exposition, we make a few mild assumptions (the weaknesses of the transforms remain without these assumptions). First let $IV = 0^n$ and assume that $\mathsf{pad}(M)$ always returns M when $|M|$ is a multiple of d. We utilize the collision-resistant hash function h, defined in Figure 3, that maps 0^{d+n} to 0^n. Define $X = 0^d \,\|\, 1^d$ and $Y = 0^{2d} \,\|\, 1^d$. To show that the four constructions are not CR-Pr we simply specify two messages X and Y which cause the hash function to collide, but do not cause any collisions on the compression function. Particularly, let $X = 0^d \,\|\, 1^d$ and $Y = 0^{2d} \,\|\, 1^d$. Then $\mathrm{PRE}^h(X) = \mathrm{PRE}^h(Y)$, and since neither message is a prefix of the other this suffices to show that PRE is not CR-Pr. For the other three constructions, we have that $X = 0^d$ and $Y = 0^{2d}$ collide. Never do any of these cause a collision against h.

The straightforward counter-examples exploit the weakness of the basic MD transform. As noted previously, the MD transform does not give any guarantees about collision resistance, and only when we consider particular padding functions (i.e., $\mathsf{pad64}$) can we create a CR-Pr transform. Likewise, we have illustrated that the mechanisms of prefix-free encodings, dropping output bits, and enveloping do nothing to help ensure collision-resistance is preserved, even though they render the transforms PRO-Pr. To properly ensure preservation of both properties, we must specify transforms that make use of mechanisms that ensure collision-resistance preservation *and* mechanisms that ensure pseudorandom oracle preservation. In fact, it is likely that adding strengthening to these transforms would render them CR-Pr. However, as we show in the next section, our new construction (with strengthening) is already more efficient than these constructions (without strengthening).

PRF preservation. It is not formally meaningful to compare PRF preservation with PRO preservation, since the resulting schemes in either case are different types of objects (one keyed and one keyless). However we can look at particular transforms. Of the four proposed by Coron et al. only PRE is known to be PRF-Pr. Let e be a PRF. Since we are using the key-via-IV strategy, the keyed version of PRE^e is $\mathrm{PRE}^e_K(M) = e^+(K, \mathsf{padPF}(M))$. This is already known to be a good PRF [1]. As for the other three transforms, it is unclear whether any of them are PRF-Pr. For NT, we note that the security will depend greatly on the assumptions made about g. For example, if g is not one-way, then an adversary can determine the values produced by the underlying MD iteration and mount simple length-extension attacks. Instead of analyzing these transforms further (which are not CR-Pr anyway), we look at a new construction.

5 A Transform that Preserves all Three Properties: EMD

We propose a transform that is both CR-Pr, PRO-Pr, and PRF-Pr. Let n, d be numbers such that $d \geq n + 64$. Let c: $\{0,1\}^{d+n} \to \{0,1\}^n$ be a function and let $D^\circ = \cup_{i \geq 1}\{0,1\}^{(i+1)d-n}$. Then we define the enveloped Merkle-Damgård iteration c°: $\{0,1\}^{2n} \times D^\circ \to \{0,1\}^n$ on c by the algorithm given in Figure 5.

To specify our transform we require a padding

Algorithm $c^{\circ}(I_1, I_2, M)$:
$M_1 \cdots M_k \xleftarrow{d} M$
$P \leftarrow M_1 \cdots M_{k-1}$
return $c(c^{+}(I_1, P) \parallel M_k \parallel I_2)$

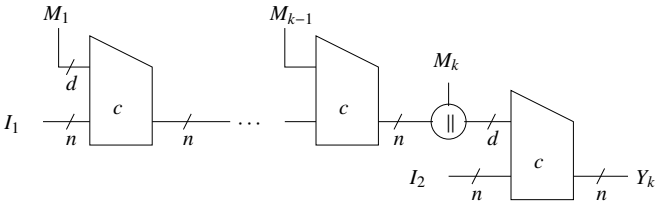


Figure 5: The EMD iteration.

function padEMD: $\{0,1\}^{\leq 2^{64}} \to D^{\circ}$ for which the last 64 bits of padEMD(M) encodes $\langle |M| \rangle_d$. Fix $IV1, IV2 \in \{0,1\}^n$ with $IV1 \neq IV2$. Then we specify the *enveloped Merkle-Damgård transform* EMD$[c] = c^{\circ}(IV1, IV2, \text{padEMD}(\cdot))$.

The new transform utilizes two main mechanisms for ensuring property preservation. The first is the well-known technique of strengthening: we require a padding function that returns a string appended with the 64-bit encoding of the length. This provides domain separation for strings of distinct lengths, which particularly ensures that EMD preserves collision-resistance. The second technique is using an 'extra' compression function application to envelope the internal MD iteration. Our envelope technique differs from prior work in that we allow message bits to be included in the envelope's input (previously, these bits would be a fixed constant, out of adversarial control). This results in a performance improvement since in practice it is always desirable to have d as large as possible relative to n (e.g., in SHA-1 $d = 512$ and $n = 160$). Finally, we utilize two distinct initialization vectors to provide (with high probability) domain separation between the envelope and internal applications of the compression function. This mechanism, and the associated proof that justifies its use (see the proof of Theorem 2), allows us to dispel with more costly domain separation techniques such as prefix-free encodings or the prepending of 0^d to messages (as used in the HMAC Transform).

5.1 Security of EMD

Collision-resistance preservation. Let h: $\{0,1\}^{d+n} \to \{0,1\}^n$ be a collision resistant hash function. Then any adversary which finds collisions against EMD^h (two messages $M \neq M'$ for which EMD$^h(M) =$ EMD$^h(M')$) will necessarily find collisions against h. This can be proven using a slightly modified version of the proof that SMD is collision-resistant [12, 10], and we therefore omit the details. The important intuition here is that embedding the length of messages in the last block is crucial; without the strengthening the scheme would *not* be collision resistant (similar attacks as those given in Section 4 would be possible).

PRO preservation. Now we show that EMD is PRO-Pr. We first prove a slightly different transform is PRO-Pr and then show that EMD reduces to this other transform. Let $f, g = \mathsf{RF}_{d+n,n}$ be random oracles. For any strings $P_1 \in D^{+}$ and $P_2 \in \{0,1\}^{d-n}$ we define the function gf^{+}: $D^{\circ} \to \{0,1\}^n$ by $gf^{+}(P \parallel S) = g(f^{+}(IV1, P_1) \parallel P_2 \parallel IV2)$. This function is essentially EMD^f, except that we replace the envelope with an independent random oracle g. The following lemma states that gf^{+} is a pseudorandom oracle.

Lemma 1 [gf^{+} is a PRO] *Let $f, g = \mathsf{RF}_{d+n,n}$. Let A be an adversary that asks at most q_L left queries, q_f right f-queries, q_g right g-queries and runs in time t. Then*

$$\mathbf{Adv}^{\text{pro}}_{gf^{+},SB}(A) \leq \frac{(q_L + q_g)^2 + q_f^2 + q_g q_f}{2^n}$$

where $SB = (SB_f, SB_g)$ is defined in Figure 6 and $q_{SB} \leq q_g$ and $t_{SB} = O(q_f^2 + q_g q_f)$.

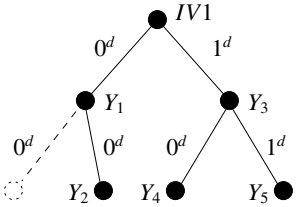
We might hope that this result is given by Theorem 4 from [9], which states that NT^{f,g} is indifferentiable from a random oracle. Unfortunately, their theorem statement does not allow for adversarially-specified bits included in the input to g. Thus we give a full proof of Lemma 1. Here we give some intuition regarding the proof and the simulator; the full proof and more details are given in a full version of this paper, see [14].

The simulator $SB = (SB_f, SB_g)$ exposes two interfaces that accept $(n + d)$-bit inputs and reply with n bit outputs. Its goal is to behave in such a way that no adversary can determine (with high probability) that it is not dealing with the construction and two random oracles f and g. The first interface mimics the internal random oracle f and the second mimics the enveloping random oracle g. The simulator maintains a tree structure that stores information about adversarial queries (the edges) and the replies given (the nodes). The root is labeled with $IV1$. The tree below is an example after several queries. For example, a query $SB_f(0^d \parallel IV1)$ adds the left child of the root; the random value Y_1 is returned to the adversary. If the next query is $SB_f(0^d \parallel Y_1)$, then the simulator associates these two queries by producing the right child of Y_1,

```
On query SB_f(X):
    Y ←$ {0,1}^n
    Parse X into U || V s.t. |U| = d, |V| = n
    if V = IV1 then NewNode(U) ← Y
    if M_1 ··· M_i ← GetNode(V) then
        NewNode(M_1 ··· M_i U) ← Y
    return Y
On query SB_g(X):
    Parse X into V || U || W s.t. |V| = n, |U| = d − n, |W| = n
    if W = IV2 and M_1 ··· M_i ← GetNode(V) then
        return F(M_1 ··· M_i U)
    return Y ←$ {0,1}^n
```

```
On query SA(X):
    Y ←$ {0,1}^n
    Parse X into V || U || W s.t. |V| = n, |U| = d − n, |W| = n
    if W = IV2 then
        if M_1 ··· M_i ← GetNode(V) then
            return F(M_1 ··· M_i U)
        else return Y
    Parse X into U || V s.t. |U| = d, |V| = n
    if V = IV1 then NewNode(U) ← Y
    if M_1 ··· M_i ← GetNode(V) then
        NewNode(M_1 ··· M_i U) ← Y
    return Y
```

Figure 6: Pseudocode for simulators *SB* (utilized in the proof of Lemma 1) and *SA* (utilized in the proof of Theorem 2).



labeled accordingly. The dotted-out left child would correspond to a pointless query, and thus can't appear in the tree. Finally, if the adversary queries $SB_g(Y_2 \parallel M \parallel IV2)$ (for any $M \in \{0,1\}^{d-n}$) the simulator searches the tree for a node labeled Y_2, and finding one, returns $\mathcal{F}(0^d \parallel 0^d \parallel M)$ (using the edge labels on the path from the root to form this query). Note that if the low bits are not $IV2$, the simulator just returns random bits. Intuitively, the simulator will succeed whenever no Y values collide and the adversary does not predict a Y value.

The next theorem captures the main result.

Theorem 2 [EMD is PRO-Pr] *Fix n, d, and let $IV1, IV2 \in \{0,1\}^n$ with $IV1 \neq IV2$. Let $f = \mathsf{RF}_{d+n,n}$ and $\mathcal{F} = \mathsf{RF}_{*,n}$ be random oracles. Let A be an adversary that asks at most q_L left queries, each of length no larger than ld bits, q_1 the number of right queries with lowest n bits not equal to $IV2$, q_2 the number of right queries with lowest n bits equal to $IV2$, and runs in time t. Then*

$$\mathbf{Adv}^{\text{pro}}_{\text{EMD},\,SA}(A) \leq \frac{(q_L + q_2)^2 + q_1^2 + q_2 q_1}{2^n} + \frac{l q_L^2}{2^n}\,.$$

where the simulator SA is defined in Figure 6 and $q_{SA} \leq q_2$ and $t_{SA} = O(q_1^2 + q_2 q_1)$.

Proof: Let $f = \mathsf{RF}_{d+n,n}$. Note that EMD^f is just a special case (due to padding) of the function $\mathbf{f}^\circ(M) = f^\circ(IV1, IV2, M)$ and we therefore prove the more general function is a PRO. Let $\mathcal{F} = \mathsf{RF}_{D^\circ,n}$. In Figure 6 we define the simulator SA whose job is to mimic f in a way that convinces any adversary that $\mathcal{F}$ is actually $\mathbf{f}^\circ$. The behavior of SA is essentially identical to SB, the only difference is that we use $IV2$ to distinguish the envelope from internal applications of f.

Let A be an adversary attempting to differentiate between $\mathbf{f}^\circ, f$ and $\mathcal{F}, SA^{\mathcal{F}}$. We will show how this adversary can be used to construct a pro-adversary B against gf^+ (i.e., one that attempts to distinguish between gf^+, f, g and $\mathcal{F}, SB_{\mathcal{F}}, SB_g$). We utilize the three games shown in Figure 7 to perform the reduction. The first game G0 simulates exactly the pair of oracles $\mathbf{f}^\circ, f$. It uses two tables f and f_{IV2} to implement the random oracle f. The f_{IV2} table is used to track all domain points for which the low n-bits are equal to $IV2$. The f table tracks the other domain points. Note that the f table also can have domain/range pairs defined for domain points with the low bits equal to $IV2$, however these are never used in the rest of the game. We thus have that $\Pr\ A^{\mathbf{f}^\circ, f} \Rightarrow 1\ = \Pr\ A^{\text{G0}} \Rightarrow 1\ .$ We create a new game G1 (the second figure with the boxed statement included) by splitting the right oracle of G0 into two oracles: one for accessing the f table and one for accessing the f_{IV2} table. Additionally we add a flag *bad*, set to true at line 004. This game now reveals three interfaces, and so we create a new adversary B that behaves exactly as A except as follows. Whenever A queries its right oracle on a string X, we have B query $\mathsf{R}_f(X)$ if the low n bits of X are not $IV2$. Otherwise B queries $\mathsf{R}_{f_{IV2}}(X)$. Because G1 returns values to B that are distributed identically to those G0 returns to A we have that $\Pr\ A^{\text{G0}} \Rightarrow 1\ = \Pr\ B^{\text{G1}} \Rightarrow 1\ .$ Our final game is G2 (second figure with the boxed statement removed). By removing line 005, the new game G2 separates the single random oracle in the prior games into two separate random oracles. Since G1 and G2 are identical until bad we have that $\Pr\ B^{\text{G1}} \Rightarrow 1\ - \Pr\ B^{\text{G2}} \Rightarrow 1\ \leq \Pr\ B^{\text{G2}}$ sets *bad* . The right hand side of this equation can be upper bound as follows. The total number of times that line 003 can be executed is lq_L. Each time a (potentially) different random value Y_{i-1} is tested for equality against a fixed constant $IV2$. Thus we have that $\Pr\ B^{\text{G2}}$ sets *bad* $\leq lq_L/2^n$.

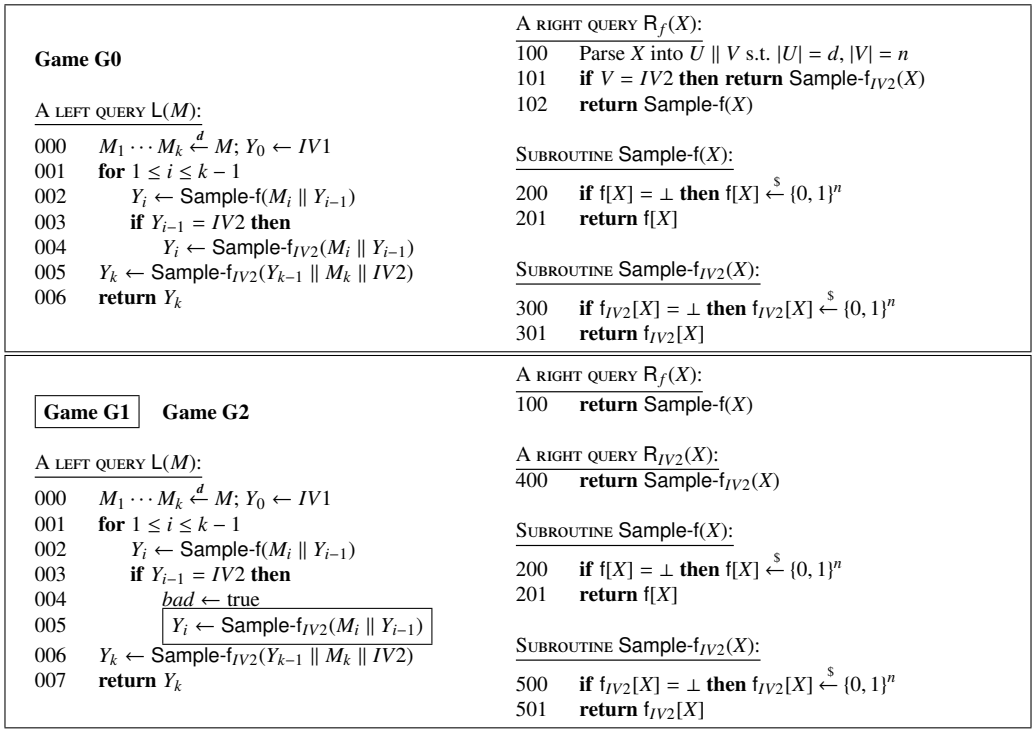


Figure 7: Games utilized in proof of Theorem 2.

Now we argue that $\Pr\ A^{\mathcal{F},SA} \Rightarrow 1\ = \Pr\ B^{\mathcal{F},SB_f,SB_g} \Rightarrow 1$. Referring back to Figure 6, we see that SA and SB behave identically if all queries to SB_g from an adversary have the low n bits equal to $IV2$ and all queries to SB_f from an adversary have the low n bits not equal to $IV2$. But this is the behavior of B, by construction, and so the probabilities are equal. Combining all of the above facts we get that

$$\begin{aligned}
\mathbf{Adv}^{\text{pro}}_{\text{EMD}^f,SA}(A) &\le \mathbf{Adv}^{\text{pro}}_{\mathbf{f}^\circ,SA}(A) \\
&= \Pr\ A^{\mathbf{f}^\circ,f} \Rightarrow 1\ - \Pr\ A^{\mathcal{F},SA} \Rightarrow 1 \\
&= \Pr\ A^{\text{G0}} \Rightarrow 1\ - \Pr\ A^{\mathcal{F},SA} \Rightarrow 1 \\
&= \Pr\ B^{\text{G1}} \Rightarrow 1\ - \Pr\ B^{\mathcal{F},SB_f,SB_g} \Rightarrow 1 \\
&\le \Pr\ B^{\text{G2}} \Rightarrow 1\ + \frac{lq_L}{2^n} - \\
&\quad \Pr\ B^{\mathcal{F},SB_f,SB_g} \Rightarrow 1 \\
&= \mathbf{Adv}^{\text{pro}}_{gf^+,SB}(B) + \frac{lq_L}{2^n}\,.
\end{aligned}$$

Let q_1 be the number of queries by B to SB_f and q_2 be the number of queries to SB_g. Then we apply Lemma 1 and the theorem statement follows. ∎

PRF PRESERVATION. We utilize the key-via-IV strategy to create a keyed version of our transform, which is $\text{EMD}^e_{K_1,K_2}(M) = e^\circ(K_1, K_2, M)$ (for some PRF e). The resulting scheme is very similar to NMAC, which we know to be PRF-Pr [2]. Because our transform allows direct adversarial control over a portion of the input to the envelope function, we can not can not directly utilize the proof of NMAC (which assumes instead that these bits are fixed constants). However, the majority of the proof of NMAC is captured by two lemmas. The first (Lemma 3.1 [2]) shows (informally) that the keyed MD iteration is unlikely to have outputs that collide. The second lemma (Lemma 3.2 [2]) shows that composing the keyed MD iteration with a separately keyed PRF yields a PRF. We omit the details.

Theorem 3 [EMD is a PRF-Pr] *Fix n, d and let e: $\{0,1\}^{d+n} \to \{0,1\}^n$ be a function family keyed via the low n bits of its input. Let A be a prf-adversary against keyed* EMD *using q queries of length at most m blocks and running in time t. Then there exists prf-adversaries A_1 and A_2 against e such that*

$$\mathbf{Adv}^{\text{prf}}_{\text{EMD}^e_{K_1,K_2}}(A) \le \mathbf{Adv}^{\text{prf}}_e(A_1) + \binom{q}{2}\left[2m \cdot \mathbf{Adv}^{\text{prf}}_e(A_2) + \frac{1}{2^n}\right]$$

where A_1 utilizes q queries and runs in time at most t and A_2 utilizes at most two oracle queries and runs in time $O(mT_e)$ where T_e is the time for one computation of e.

References

[1] M. Bellare, R. Canetti, and H. Krawczyk. Pseudorandom functions revisited: the cascade construction and its concrete security. In *FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, page 514, Washington, DC, USA, 1996. IEEE Computer Society.

[2] Mihir Bellare. New Proofs for NMAC and HMAC: Security Without Collision-Resistance. Cryptology ePrint Archive, Report 2006/043, 2006. `http://eprint.iacr.org/`.

[3] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, pages 1–15, London, UK, 1996. Springer-Verlag.

[4] Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *CCS '93: Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73, New York, NY, USA, 1993. ACM Press.

[5] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In *EUROCRYPT*, pages 92–111, 1994.

[6] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with rsa and rabin.5. In *EUROCRYPT*, pages 399–416, 1996.

[7] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT*, pages 409–426, 2006.

[8] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[9] Jean-Sebastien Coron, Yevgeniy Dodis, Cecile Malinaud, and Prashant Puniya. Merkle-Damgard Revisited: How to Construct a Hash Function. In *Advances in Cryptology - Crypto'05, Lecture Notes in Computer Science*, volume 3621, pages 21–39, London, UK, 2004. Springer-Verlag.

[10] Ivan Bjerre Damgård. A design principle for hash functions. In *CRYPTO '89: Proceedings on Advances in cryptology*, pages 416–427, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[11] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *TCC*, pages 21–39, 2004.

[12] Ralph C. Merkle. One way hash functions and DES. In *CRYPTO '89: Proceedings on Advances in cryptology*, pages 428–446, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[13] Mihir Bellare and Alexandra Boldyreva and Adriana Palacio. An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem. In C. Cachin and J. Camenisch, editor, *Advances in Cryptology - EUROCRYPT'04, Lecture Notes in Computer Science*, volume 3027. Springer-Verlag, 2004.

[14] Mihir Bellare and Thomas Ristenpart. Multi-Property-Preserving Hash Domain Extension: The EMD Transform (full version). `http://www.cse.ucsd.edu/~tristenp/emd.html`.

[15] National Institute of Standards and Technology. *FIPS PUB 180-1: Secure Hash Standard*. April 1995. Supersedes FIPS PUB 180 1993 May 11.

[16] RSA Laboratories. RSA PKCS #1 v2.1: RSA Cryptography Standards. `ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf`.

[17] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In *Advances in Cryptology - EUROCRYPT'05, Lecture Notes in Computer Science*, pages 19–35, 2005.

[18] Xiaoyun Wang and Yiqun Lisa Yin and Hongbo Yu. Finding Collisions in the Full SHA-1. In *Advances in Cryptology - CRYPTO'05, Lecture Notes in Computer Science*, pages 17–36, 2005.