A preliminary version of this paper appears in *Advances in Cryptology - ASIACRYPT 2006*, Lecture Notes in Computer Science Vol. 4284, pp. 299-314, X. Lai and K. Chen eds., Springer-Verlag, 2006. This is the full version.

# Multi-Property-Preserving Hash Domain Extension and the EMD Transform

Mihir Bellare* Thomas Ristenpart†

September 2006

**Abstract**

We point out that the seemingly strong *pseudorandom oracle preserving* (PRO-Pr) property of hash function domain-extension transforms defined and implemented by Coron et. al. [12] can actually *weaken* our guarantees on the hash function, in particular producing a hash function that fails to be even collision-resistant (CR) even though the compression function to which the transform is applied is CR. Not only is this true in general, but we show that *all* the transforms presented in [12] have this weakness. We suggest that the appropriate goal of a domain extension transform for the next generation of hash functions is to be multi-property preserving, namely that one should have a *single* transform that is simultaneously at least collision-resistance preserving, pseudorandom function preserving and PRO-Pr. We present an efficient new transform that is proven to be multi-property preserving in this sense.

**Keywords:** Hash functions, random oracle, Merkle-Damgård, collision-resistance, pseudorandom function.

*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`. 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`. URL: `http://www-cse.ucsd.edu/users/tristenp`. Supported in part by above-mentioned grants of first author.

# Contents

## 1 Introduction

Background. Recall that hash functions are built in two steps. First, one designs a compression function $h\colon \{0,1\}^{d+n} \rightarrow \{0,1\}^n$, 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 $h$ as a black box to implement the hash function $H^h\colon \{0,1\}^* \rightarrow \{0,1\}^n$ associated to $h$. All widely-used hash functions use the Merkle-Damgård (MD) transform [16, 13] because it has been proven [16, 13] to be *collision-resistance preserving* ($\mathsf{CR\text{-}Pr}$): if $h$ is collision-resistant ($\mathsf{CR}$) 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 $\mathsf{CR}$ no longer suffices as the security goal for hash functions. In order to obtain $\mathsf{MAC}$s and $\mathsf{PRF}$s, hash functions were keyed. The canonical construct in this domain is HMAC [4, 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 [6] in public-key schemes such as RSA-OAEP [7] and RSA-PSS [8] in the RSA PKCS#1 v2.1 standard [18]. $\mathsf{CR}$ is insufficient for arguing the security of hash function based $\mathsf{MAC}$s or $\mathsf{PRF}$s, 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-$\mathsf{CR}$ 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 $\mathsf{CR}$.

This paper. Towards the goal of building strong, multi-purpose hash functions, our focus is on domain extension, meaning we wish to determine which domain extension transforms are best suited to this task. The first part of our work examines a natural candidate, namely transforms that are *pseudorandom oracle preserving* as per [12], and identifies some weaknesses of this goal. This motivates the second part, where we introduce the notion of a *multi-property preserving (MPP) transform*, argue that this should be the target goal, and present and prove the correctness of an efficient MPP transform that we refer to as EMD. Let us now look at all this in more depth.

Random-oracle preservation. Coron, Dodis, Malinaud and Puniya [12] make the important observation that random oracles are modeled as monolithic entities (i.e., are black boxes working on domain $\{0,1\}^*$), but in practice are instantiated by hash functions that are highly structured due to the design paradigm described above, leading for example to the extension attack. Their remedy for this logical gap is to suggest that a transform $H$ be judged secure if, when modeling $h$ as a fixed-input-length random oracle, the resulting scheme $H^h$ behaves like a random oracle. They give a formal definition of "behaving like a random oracle" using the indifferentiability framework of Maurer et al. [14]. We use the moniker *pseudorandom oracle* to describe any construction that is indifferentiable from a random oracle. (Note that a random oracle itself is always a pseudorandom oracle.) The framework has the desirable property that any scheme proven secure in the random oracle model of [6] is still secure when we replace the random oracles with pseudorandom oracles. We call the new security goal of [12] *pseudorandom oracle preservation* ($\mathsf{PRO\text{-}Pr}$). They propose four transforms which they prove to be $\mathsf{PRO\text{-}Pr}$.

$\mathsf{PRO\text{-}Pr}$ seems like a very strong property to have. One reason one might think this is that it appears to automatically guarantee that the constructed hash function has many nice properties. For example, that the hash function created by a $\mathsf{PRO\text{-}Pr}$ transform would be $\mathsf{CR}$. Also that the hash function could be keyed in almost any reasonable way to yield a $\mathsf{PRF}$ and $\mathsf{MAC}$. And so on. This would be true, because random oracles have these properties, and hence so do pseudorandom oracles. Thus, one is lead to think that one can stop with PRO-Pr: once the transform has this property, we have all the attributes we desire from the constructed hash function.

WEAKNESS OF PRO-Pr. The first contribution of this paper is to 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 [11, 3] about the sometimes impossibility of instantiating random oracles.

THE TRANSFORMS OF [12] ARE NOT CR-Pr. 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 [12]. 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. Unfortunately, we show that none of the four transforms is CR-Pr. We do this by presenting an example CR compression function $h$ such that applying each of the four transforms to it results in a hash function for which finding collisions is trivial. 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.

WHAT THIS MEANS. We clarify that we are *not* suggesting that the pseudorandom oracle preservation goal of [12] 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, however, is that *by itself*, it is not enough because it can weaken existing, standard-model guarantees. This leads to the question of what exactly *is* enough, or what we should ask for in terms of a goal for hash domain extension transforms.

MPP TRANSFORMS. 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. We suggest that, if we want a hash function with properties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ then we should (1) design a compression function $h$ with the goal of having properties $\mathsf{P}_1, \ldots, \mathsf{P}_n$, and (2) apply a domain extension transform $H$ that *provably* preserves $\mathsf{P}_i$ 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-*

| **Transform** | **CR-Pr** | **PRO-Pr** | **PRF-Pr** | **Uses of $h$ for $\|M\| = b \geq d$** |
|---|---|---|---|---|
| Plain MD (MD) | No | No | No | $\lceil (b+1)/d \rceil$ |
| Strengthened MD (SMD) | [16, 13] | No | No | $\lceil (b+1+64)/d \rceil$ |
| Prefix-Free (PRE) | No | [12] | [5] | $\lceil (b+1)/(d-1) \rceil$ |
| Chop Solution (CHP) | No | [12] | ? | $\lceil (b+1)/d \rceil$ |
| NMAC Construction (NT) | No | [12] | ? | $1 + \lceil (b+1)/d \rceil$ |
| HMAC Construction (HT) | No | [12] | ? | $2 + \lceil (b+1)/d \rceil$ |
| Enveloped MD (EMD) | [16] | Thm. 5.2 | Thm. 5.3 | $\lceil (b+1+64+n)/d \rceil$ |

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

---

*preserving domain extension transform* (from now on simply an MPP 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. We suggest that multi-property preservation is the goal a transform should target.

PROPERTIES TO PRESERVE. Of course the next question to ask is which properties our MPP domain extension transform should preserve. We wish, of course, that the transform continue to be CR-Pr, meaning that it preserve CR. The second thing we ask 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 goal is important due to the many uses of hash functions as MACs and PRFs via keying as mentioned above. Indeed, if we have a compression function that can be keyed to be a PRF and our transform is PRF-Pr then obtaining a PRF or MAC from a hash function will be simple and the construction easy to justify. The final goal we will ask is that the transform be PRO-Pr. Compelling arguments in favor of this goal were made at length in [12] and briefly recalled above.

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. We note that such a MPP transform will not suffer from the weakness of the transforms of [12] noted above because it will be not only PRO-Pr but also CR-Pr and PRF-Pr.

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 combines two mechanisms to ensure that it preserves all the properties of interest. The first mechanism is the well-known Merkle-Damgård strengthening [16]: 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 — we apply the compression function in a distinguished way to the output of the plain MD iteration. Envelopes in this setting were previously used by the NMAC and HMAC constructions [4] to build PRFs out of compression functions, and again in two of the PRO-Pr transforms of [12], which were also based on NMAC and HMAC. We utilize the envelope in a way distinct from these prior constructions. Particularly, we include message bits as input to 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 simple solution allows our transform to be PRO-Pr using a single random oracle without using the other work-arounds previously suggested (e.g., prefix-free encodings or prepending a block of zeros to input messages). A comparison of various transforms is given in Fig. 1.

Patching existing transforms. We remark that it is possible to patch the transforms of [12] 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 [5] implies that one of the transforms of [12] is PRF-Pr, but whether or not this is true for the others is not clear.

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. This assumption might seem questionable in light of the recent collision-finding attacks [19, 20] that have destroyed some hash functions and tainted others. But we recall that for block ciphers, 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 that the planned NIST hash function competition will lead to compression functions having the properties (CR and beyond) that we want, and perhaps without increase, or even with decrease, in cost, compared to current compression functions. We also note that we are not really making new requirements on the compression function; we are only making explicit requirements that are implicit even in current usage.

Families of compression functions. Several works [1, 9, 15] consider a setting where compression and hash functions are families rather than individual functions, meaning, like block ciphers, have an extra, dedicated key input. In contrast, we, following [4, 12, 2], adopt the setting of current practical cryptographic compression and hash functions where there is no such dedicated key input. An enveloping technique similar to that of EMD is used in the Chain-Shift construction of Maurer and Sjödin [15] for building a VIL MAC out of a FIL MAC in the dedicated key input setting. We further discuss this setting, and their work, in Appendix A.

## 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 $M_1$, $M_2$, …, $M_k$ where $|M_i| = d$ for $1 \leq i \leq k$, otherwise parse $M$ into $M_1$, $M_2$, …, $M_{k-1}$, $M_k$ 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 $\mathsf{M}^{\mathsf{N}}$ to signify that $\mathsf{M}$ gets to query all the interfaces of $\mathsf{N}$. For a set *Dom* and finite set *Rng* we define a *random function* by the following TM accepting inputs $X \in \mathit{Dom}$:

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

where $T$ is a table everywhere initialized to $\bot$. 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 $\mathit{Dom} = \{0,1\}^d$ and $\mathit{Rng} = \{0,1\}^r$ we write $\mathsf{RF}_{d,r}$ in place of $\mathsf{RF}_{\mathit{Dom},\mathit{Rng}}$. We similarly define $\mathsf{RF}_{d,\mathit{Rng}}$ and $\mathsf{RF}_{\mathit{Dom},r}$ in the obvious ways and write $\mathsf{RF}_{*,r}$ in the special case that $\mathit{Dom} = \{0,1\}^*$. A *random oracle* is simply a public random function: all parties (including the adversary) are given access. We write $f, g, \ldots = \mathsf{RF}_{\mathit{Dom},\mathit{Rng}}$ to signify that $f$, $g$, $\ldots$ are independent random oracles from *Dom* to *Rng*. 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 (CR) 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 compression function with signature $h\colon\ \{0,1\}^{d+n} \to \{0,1\}^n$.

Note our definition of CR is informal. The general understanding in the literature is that a formal treatment requires considering keyed families. But practical compression and hash functions are not keyed when used for CR. (They can be keyed for use as MACs or PRFs.) And in fact, our results on CR are still formally meaningful because they specify explicit reductions.

PRFs. Let $F\colon\ \mathit{Keys} \times \mathit{Dom} \to \mathit{Rng}$ 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 $K \xleftarrow{\$} \mathit{Keys}$ and a random function $f = \mathsf{RF}_{\mathit{Dom},\mathit{Rng}}$. More compactly we write the *prf-advantage* of an adversary $A$ as

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

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 $e\colon\ \{0,1\}^{d+n} \to \{0,1\}^n$ that is keyed through the low $n$ bits of the input.

PROs. The indifferentiability framework [14] 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 $C^{f_1,\ldots,f_l}\colon\ \mathit{Dom} \to \mathit{Rng}$ 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 $\mathcal{F} = \mathsf{RF}_{\mathit{Dom},\mathit{Rng}}$ 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

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

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 $\mathsf{RF}_{d+n,n}$.

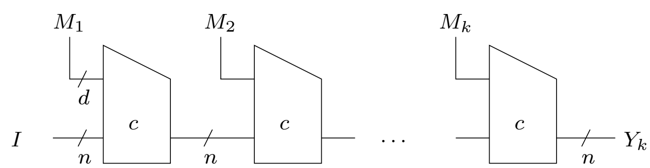
Resources. We give concrete statements about the advantage of adversaries using certain resources. For prf-adversaries we measure the total number of queries $q$ made and the running time $t$. For pro-adversaries we measure the total number of *left queries* $q_L$ (which are either to $C$ or $\mathcal{F}$) 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 adversaries 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. Let $c$: $\{0,1\}^{d+n} \to \{0,1\}^n$ be an arbitrary fixed-input-length function. Using it, 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 below.

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




We will also write $f^+$, $h^+$, and $e^+$ which are defined just like $c^+$ but with $c$ replaced with the appropriate function. Since $I$ is usually fixed to a constant, the function $c^+$ only works for strings that are a multiple of $d$ bits. Thus we require a padding function $\mathsf{pad}(M)$, which for any string $M \in \{0,1\}^*$ returns a string $Y$ for which $|Y|$ is a multiple of $d$. We require that $\mathsf{pad}$ is one-to-one (this requirement is made for all padding functions in this paper). A standard instantiation for $\mathsf{pad}$ is to append to the message a one bit and then enough zero bits to fill out a block. 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 $\mathrm{MD}^e$ 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 $C_K^e$ is a PRF. A transform for which all of the above holds is considered *multi-property preserving.*

SECURITY OF MD AND SMD. It is well known that MD is neither CR-Pr, PRO-Pr, or PRF-Pr [16, 13, 5, 12]. 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 [17], 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 $kd$ bits for some number $k \geq 1$ 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 [5, 12].

# 4 Orthogonality of Property Preservation

In this section we illustrate that property preservation is orthogonal. Previous work [12] 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 specific PRO-Pr transforms. Thus, we investigate the four transforms 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 to $0^n$ 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. On the other hand, we now show how

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

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

$\boxed{\text{Game G0}}$ Game G1

**procedure Initialize**
$f = \mathsf{RF}_{d+n,n}$

**procedure** $f(x)$
**ret** $f(x)$

**procedure** $C(X)$
$Y \leftarrow C_1^f(X)$
**if** $f(0^{d+n}) = 0^n$ **then** $bad \leftarrow \mathsf{true}$; $\boxed{Y \leftarrow 0^n}$
**ret** $Y$

Figure 2: (Top) Definition of the collision-resistant compression function used in the proof of Proposition 4.1 and the counter-examples of Section 4.2. (Bottom) Games utilized in the proof of Proposition 4.1 to show that $C_2^f$ is a $\mathsf{PRO}$.

to easily design a collision-resistant hash function $h$ that causes $C_2^h$ to not be collision resistant. Let $h'\colon \{0,1\}^{d+n} \to \{0,1\}^{n-1}$ be some collision-resistant hash function. Then $h(M)$ returns $0^n$ if $M = 0^{d+n}$, otherwise it returns $h'(M) \,\|\, 1$. 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 4.1** [$C_2$ **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 $A$ that utilizes $q_L$ left queries, $q_R$ right queries, and runs in time $t$, we have that*

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

**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 2 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 [10] that $\Pr[A^{\mathrm{G0}} \Rightarrow 1] - \Pr[A^{\mathrm{G1}} \Rightarrow 1] \le \Pr[A^{\mathrm{G1}} \text{ sets } \mathsf{bad}]$. The right hand side is equal to $2^{-n}$ because $f$ is a random oracle. Thus,

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

■

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

Figure 3: The four MD variants proposed in [12] that are PRO-Pr but not CR-Pr.

## 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 a 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 Fig. 3 and below, see [12] 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 $\mathrm{NT}[c, g] = g(c^{+}(IV, \mathsf{pad}(\cdot)))$, where $g$ is a function from $n$ bits to $n$ bits (distinct from $h$). Lastly, the *HMAC Transform* is defined by $\mathrm{HT}[c] = c(c^{+}(IV, 0^d \mid\mid \mathsf{pad}(\cdot)) \mid\mid 0^{d-n} \mid\mid IV)$. 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.

Let $IV = 0^n$. We shall use the collision-resistant hash function $h$ that maps $0^{d+n}$ to $0^n$ (defined in Sect. 4.1). We first show that the PRE construction, while being PRO-Pr for all prefix-free encodings, is not CR-Pr for all prefix-free encodings. Let $\mathsf{padPF}(M) = g_2(M)$ from Sect. 3.3 of [12]. Briefly, $g_2(M) = 0 \mid\mid M_1, \ldots, 0 \mid\mid M_{k-1}, 1 \mid\mid M_k$ for $M_1 \mid\mid \cdots \mid\mid M_k \stackrel{d-1}{\leftarrow} M \mid\mid 10^r$, where $r = (d-1) - ((|M|+1) \bmod d-1)$. (That is we append a one to $M$, and then enough zero's to make a string with length a multiple of $d-1$.) Now let $X = 0^{d-1}$ and $Y = 0^{2(d-1)}$. Then we have that $\mathrm{PRE}^h(X) = \mathrm{PRE}^h(Y)$ and no collisions against $h$ occur. We should note that *some* prefix-free encodings will render PRE CR-Pr, for example any that also include strengthening. The important point here is that strengthening does not ensure prefix-freeness and vice-versa.

For the other three constructions, we assume that $\mathsf{pad}(M)$ simply appends a one and then enough zeros to make a string with length a multiple of $d$. Now let $X = 0^d$ and $Y = 0^{2d}$. Then we have that $\mathrm{CHP}^h(X) = \mathrm{CHP}^h(Y)$ and $\mathrm{NT}^h(X) = \mathrm{NT}^h(Y)$ and $\mathrm{HT}^h(X) = \mathrm{HT}^h(Y)$. Never is there a collision generated against $h$.

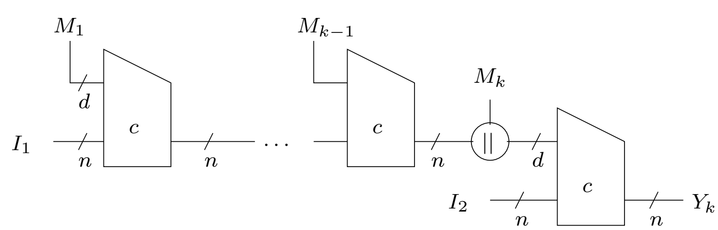
The straightforward counter-examples exploit the weakness of the basic MD transform. As noted previously, the MD transform does not give any guarantees about collision resistance, and only when we consider particular padding functions (i.e., pad64) can we create a CR-Pr transform. Likewise, we have illustrated 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 $\mathrm{PRE}^e$ is $\mathrm{PRE}^e_K(M) = e^+(K, \mathsf{padPF}(M))$. This is already known to be a good PRF [5]. 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 $g$. If $g$ is a separately keyed PRF, then we can apply the proof of NMAC [4]. On the other hand, 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 The EMD Transform

We propose a transform that is 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 $c^\circ$: $\{0,1\}^n \times \{0,1\}^n \times D^\circ \to \{0,1\}^n$ on $c$ by the algorithm given below.

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




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

EMD 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 ensures that EMD preserves collision-resistance. The second technique consists of using an 'extra' compression function application to envelope the internal MD iteration. It is like the enveloping mechanism used by Maurer and Sjödin in a different setting [15] (which is discussed in more detail in Appendix A), but distinct from prior enveloping techniques used in the current setting. First, it includes message bits in the envelope's input (in NMAC/HMAC and HT, 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 $n = 160$). Second, it utilizes a distinct initialization vector to provide (with high probability) domain separation between the envelope and internal applications of the compression function. This mechanism allows us to avoid having to use other previously proposed domain separation techniques while still yielding a PRO-Pr transform. (The previous techniques were prefix-free encodings or the prepending of $0^d$ to messages, as used in the HT transform; both are more costly.)

### 5.1 EMD is CR-Pr

Let $h$: $\{0,1\}^{d+n} \to \{0,1\}^n$ be a collision resistant hash function. Then any adversary which finds collisions against $\mathrm{EMD}^h$ (two messages $M \neq M'$ for which $\mathrm{EMD}^h(M) = \mathrm{EMD}^h(M')$) will necessarily find collisions against $h$. This can be proven using a slightly modified version of the proof that SMD is collision-resistant [16, 13], 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).

### 5.2 EMD is PRO-Pr

Now we show that EMD is PRO-Pr. We proceed in two steps. First we state and discuss a lemma that is core to the proof that EMD is PRO-Pr. Particularly, this lemma proves that a slightly different transform is PRO-Pr. Second, we use this lemma in the proof of our main result, captured by Theorem 5.2. That proof boils down to showing that EMD behaves indifferentiably from this other, simplifed transform, and then by applying the lemma we finish. The proof of the lemma is lengthy, and differed to Section 6.

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 $gf^+(P_1 \,||\, P_2) = g(f^+(IV1, P_1) \,||\, P_2 \,||\, IV2)$. This function is essentially $\mathrm{EMD}^f$, except that we replace the envelope with an independent random oracle $g$. The following lemma states that $gf^+$ is a pseudorandom oracle.

**Lemma 5.1** [$gf^+$ **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*

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

*where $SB = (SB_f, SB_g)$, defined in Fig. 4, makes $q_{SB} \leq q_g$ queries and runs in time $\mathcal{O}(q_f^2 + q_g q_f)$.*

We might hope that this result is given by Theorem 4 from [12], which states that $\mathrm{NT}^{f,g}$ is indifferentiable from a random oracle. Unfortunately, their theorem statement does not allow for adversarially-specified bits included in the input to $g$. The proofs are likely similar, however since no proof of the Coron et al. theorem has actually appeared we were forced to work from scratch. We give the full proof of the lemma in Section 6. Now we describe the simulator used, which is needed for the proof of the main result that EMD is PRO-Pr.

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 notation $\textsc{NewNode}(M_1 \cdots M_i U) \leftarrow Y$ for $U \in \{0,1\}^d$, $Y \in \{0,1\}^n$, and $M_i \in \{\varepsilon\} \cup \{0,1\}^d$ means (1) locate the node found by following the path starting from the root and following the edges labeled

**On query $SB_f(X)$:**

$Y \xleftarrow{\$} \{0,1\}^n$
Parse $X$ into $U \,||\, V$ s.t.
  $|U| = d$, $|V| = n$
**if** $V = IV1$ **then** $\textsc{NewNode}(U) \leftarrow Y$
**if** $M_1 \cdots M_i \leftarrow \textsc{GetNode}(V)$ **then**
  $\textsc{NewNode}(M_1 \cdots M_i U) \leftarrow Y$
**ret** $Y$

**On query $SB_g(X)$:**

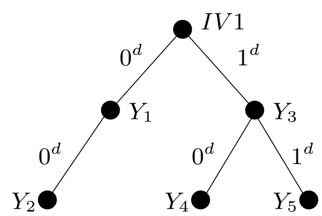
Parse $X$ into $V \,||\, U \,||\, W$ s.t.
  $|V| = n$, $|U| = d - n$, $|W| = n$
**if** $W = IV2$ **and**
    $M_1 \cdots M_i \leftarrow \textsc{GetNode}(V)$ **then**
  **ret** $\mathcal{F}(M_1 \cdots M_i U)$
**ret** $Y \xleftarrow{\$} \{0,1\}^n$

**On query $SA(X)$:**

$Y \xleftarrow{\$} \{0,1\}^n$
Parse $X$ into $V \,||\, U \,||\, W$ s.t.
  $|V| = n$, $|U| = d - n$, $|W| = n$
**if** $W = IV2$ **then**
  **if** $M_1 \cdots M_i \leftarrow \textsc{GetNode}(V)$ **then**
    **ret** $\mathcal{F}(M_1 \cdots M_i U)$
  **else**
    **ret** $Y$
Parse $X$ into $U \,||\, V$ s.t. $|U| = d$, $|V| = n$
**if** $V = IV1$ **then** $\textsc{NewNode}(U) \leftarrow Y$
**if** $M_1 \cdots M_i \leftarrow \textsc{GetNode}(V)$ **then**
  $\textsc{NewNode}(M_1 \cdots M_i U) \leftarrow Y$
**ret** $Y$

Figure 4: Pseudocode for simulators $SB$ (utilized in the proof of Lemma 5.1) and $SA$ (utilized in the proof of Theorem 5.2).

by $M_1$, $M_2$, etc. and (2) add an edge labeled by $U$ from this found node to a new node labeled by $Y$. The notation $\textsc{GetNode}(V)$ for $V \in \{0,1\}^n$ returns the sequence of edge labels on a path from the root to a node labeled by $V$ (if there are duplicate such nodes, return an arbitrary one, if there are none then return false). The tree below is an example after several queries. For example, a query $SB_f(0^d \,||\, IV1)$ adds the left child of the root; the random value $Y_1$ is returned to the adversary. If the next query is $SB_f(0^d \,||\, Y_1)$, then the simulator associates these two queries by producing the child of $Y_1$, labeled accordingly. Finally, if the adversary queries $SB_g(Y_2 \,||\, M \,||\, IV2)$ (for any $M \in \{0,1\}^{d-n}$) the simulator searches the tree for a node labeled $Y_2$, and finding one, returns $\mathcal{F}(0^d \,||\, 0^d \,||\, M)$ (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 simulator is discussed further in Section 6. Now we use Lemma 5.1 to prove that EMD is PRO-Pr.

**Theorem 5.2 [EMD is PRO-Pr]** *Fix $n$, $d$, and let $IV1, IV2 \in \{0,1\}^n$ with $IV1 \neq IV2$. Let $f = \mathsf{RF}_{d+n,n}$ be a random oracle. Let $A$ be an adversary that asks at most $q_L$ left queries (each of length no larger than $ld$ bits), $q_1$ right queries with lowest $n$ bits not equal to $IV2$, $q_2$ right queries with lowest $n$ bits equal to $IV2$, and runs in time $t$. Then*

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

*where the simulator $SA$, defined in Fig. 4, makes $q_{SA} \leq q_2$ queries and runs in time $\mathcal{O}(q_1^2 + q_2 q_1)$.*

**Proof:** Let $f = \mathsf{RF}_{d+n,n}$. Note that $\mathrm{EMD}^f$ 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 $\mathsf{PRO}$. Let $\mathcal{F} = \mathsf{RF}_{D^\circ,n}$. In Fig. 4 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 Fig. 5 to perform the reduction. The first game G0 simulates exactly the pair of oracles $\mathbf{f}^\circ, f$. It uses two tables $\mathsf{f}$ and $\mathsf{f}_{IV2}$ to implement the random oracle $f$. The $\mathsf{f}_{IV2}$ table is used to track all domain points for which the low $n$-bits are equal to $IV2$. The $\mathsf{f}$ table tracks the other domain points. Note that the $\mathsf{f}$ 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 $\Pr\left[\, A^{\mathbf{f}^\circ, f} \Rightarrow 1 \,\right] = \Pr\left[\, A^{\mathrm{G0}} \Rightarrow 1 \,\right]$. 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 $\mathsf{f}$ table and one for accessing the $\mathsf{f}_{IV2}$ 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 $\Pr\left[\, A^{\mathrm{G0}} \Rightarrow 1 \,\right] = \Pr\left[\, B^{\mathrm{G1}} \Rightarrow 1 \,\right]$. 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 $\Pr\left[\, B^{\mathrm{G1}} \Rightarrow 1 \,\right] - \Pr\left[\, B^{\mathrm{G2}} \Rightarrow 1 \,\right] \le \Pr\left[\, B^{\mathrm{G2}} \text{ sets } \mathsf{bad} \,\right]$. 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 $\Pr\left[\, B^{\mathrm{G2}} \text{ sets } \mathsf{bad} \,\right] \le lq_L/2^n$.

Now we argue that $\Pr\left[\, A^{\mathcal{F}, SA} \Rightarrow 1 \,\right] = \Pr\left[\, B^{\mathcal{F}, SB_f, SB_g} \Rightarrow 1 \,\right]$. Referring back to Fig. 4, 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

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

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 5.1 and the theorem statement follows. ▮

**Game G0**

A LEFT QUERY $\mathsf{L}(M)$:

$M_1 \cdots M_k \overset{d}{\leftarrow} M$; $Y_0 \leftarrow IV1$
**for** $1 \le i \le k-1$
  $Y_i \leftarrow \mathsf{Sample\text{-}f}(M_i \,||\, Y_{i-1})$
  **if** $Y_{i-1} = IV2$ **then**
    $Y_i \leftarrow \mathsf{Sample\text{-}f}_{IV2}(M_i \,||\, Y_{i-1})$
$Y_k \leftarrow \mathsf{Sample\text{-}f}_{IV2}(Y_{k-1} \,||\, M_k \,||\, IV2)$
**ret** $Y_k$

A RIGHT QUERY $\mathsf{R}_f(X)$:

Parse $X$ into $U \,||\, V$ s.t. $|U| = d$, $|V| = n$
**if** $V = IV2$ **then ret** $\mathsf{Sample\text{-}f}_{IV2}(X)$
**ret** $\mathsf{Sample\text{-}f}(X)$

SUBROUTINE $\mathsf{Sample\text{-}f}(X)$:

**if** $\mathsf{f}[X] = \perp$ **then** $\mathsf{f}[X] \overset{\$}{\leftarrow} \{0,1\}^n$
**ret** $\mathsf{f}[X]$

SUBROUTINE $\mathsf{Sample\text{-}f}_{IV2}(X)$:

**if** $\mathsf{f}_{IV2}[X] = \perp$ **then** $\mathsf{f}_{IV2}[X] \overset{\$}{\leftarrow} \{0,1\}^n$
**ret** $\mathsf{f}_{IV2}[X]$

$\boxed{\textbf{Game G1}}$ **Game G2**

A LEFT QUERY $\mathsf{L}(M)$:

$M_1 \cdots M_k \overset{d}{\leftarrow} M$; $Y_0 \leftarrow IV1$
**for** $1 \le i \le k-1$
  $Y_i \leftarrow \mathsf{Sample\text{-}f}(M_i \,||\, Y_{i-1})$
  **if** $Y_{i-1} = IV2$ **then**
    $bad \leftarrow \mathsf{true}$
    $\boxed{Y_i \leftarrow \mathsf{Sample\text{-}f}_{IV2}(M_i \,||\, Y_{i-1})}$
$Y_k \leftarrow \mathsf{Sample\text{-}f}_{IV2}(Y_{k-1} \,||\, M_k \,||\, IV2)$
**ret** $Y_k$

A RIGHT QUERY $\mathsf{R}_f(X)$:

**ret** $\mathsf{Sample\text{-}f}(X)$

A RIGHT QUERY $\mathsf{R}_{IV2}(X)$:

**ret** $\mathsf{Sample\text{-}f}_{IV2}(X)$

SUBROUTINE $\mathsf{Sample\text{-}f}(X)$:

**if** $\mathsf{f}[X] = \perp$ **then** $\mathsf{f}[X] \overset{\$}{\leftarrow} \{0,1\}^n$
**ret** $\mathsf{f}[X]$

SUBROUTINE $\mathsf{Sample\text{-}f}_{IV2}(X)$:

**if** $\mathsf{f}_{IV2}[X] = \perp$ **then** $\mathsf{f}_{IV2}[X] \overset{\$}{\leftarrow} \{0,1\}^n$
**ret** $\mathsf{f}_{IV2}[X]$

Figure 5: Games utilized in proof of Theorem 5.2.

### 5.3 EMD is PRF-Pr

We utilize the key-via-IV strategy to create a keyed version of our transform $\mathrm{EMD}^e_{K_1,K_2}(M) = e^\circ(K_1, K_2, M)$ (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 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 5.3 [EMD is 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*

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

*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 $\mathcal{O}(mT_e)$ where $T_e$ is the time for one computation of $e$.*

## 6 Proof of Lemma 5.1

Fix $n$ and $d$ with $d \ge n$, some $n$-bit constant $IV$, and let $f, g = \mathsf{RF}_{d+n,n}$. To simplify the proof exposition slightly, we prove that a more general version of $gf^+$ is a pseudorandom oracle, specifically $gf^+(P_1 \,||\, P_2) = g(f^+(IV, P_1) \,||\, P_2)$ for any strings $P_1 \in D^+$ and $P_2 \in D$ (i.e., we let the adversary

On query $SB_f(X)$:
$Y \xleftarrow{\$} \{0,1\}^n$
Parse $X$ into $U \,||\, V$ s.t. $|U| = d$, $|V| = n$
**if** $V = IV$ **then**
NewNode$(U) \leftarrow Y$
**if** $M_1 \cdots M_i \leftarrow$ GetNode$(V)$ **then**
NewNode$(M_1 \cdots M_i U) \leftarrow Y$
**ret** $Y$

On query $SB_g(X)$:
Parse $X$ into $V \,||\, U$ s.t. $|V| = n$, $|U| = d$
**if** $M_1 \cdots M_i \leftarrow$ GetNode$(V)$ **then**
**ret** $\mathcal{F}(M_1 \cdots M_i U)$
**ret** $Y \xleftarrow{\$} \{0,1\}^n$

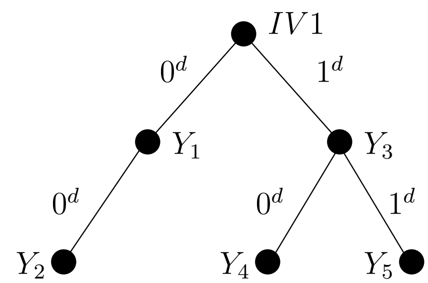



Figure 6: On the left is the simulator $SB = (SB_f, SB_g)$ which mimics the behavior of the random oracles $f$ and $g$ used by $gf^+$ (it has oracle access to a random oracle $\mathcal{F}$). The functions GetNode and NewNode are used to access and modify a tree structure, initially with only a root node labeled with $IV$. The diagram on the right is a possible tree state (not including the dotted lines, which would imply a pointless query) after several queries to $SB_f$.

also control the low $n$ bits of input into the envelope function). Thus, let $\mathcal{F} = \mathsf{RF}_{D^+,n}$. Now let $A$ be some adversary attacking the indifferentiability of $gf^+$ that asks at most $q = q_L + q_f + q_g$ queries. Recall from our definition of indifferentiability (see Section 2) that we must bound

$$\mathbf{Adv}^{\text{pro}}_{gf^+,S}(A) = \left| \Pr\left[ A^{gf^+,f,g} \Rightarrow 1 \right] - \Pr\left[ A^{\mathcal{F},SB_f,SB_g} \Rightarrow 1 \right] \right|$$

for some simulator $S$.

## 6.1 The Simulator

Figure 6 specifies the simulator $SB = (SB_f, SB_g)$, which is the same as that given in Figure 4. Here we discuss it in a bit more detail.

Recall that we do not allow pointless queries, and thus $SB_f$ and $SB_g$ need not worry about repeat queries. For queries to $SB_f$, the simulator keeps track of the input messages and the values chosen as replies, which can be done with a simple tree structure. The root is labeled with $IV$ and all other nodes are labeled with returned values. Edges are labeled with the first $d$ bits of inputs (which correspond to potential message blocks in our construction). Two subroutines are utilized to maintain our tree structure.

The notation GetNode$(V)$ performs a search of the tree, and upon finding a node with label $V$ returns the edge labels on the path from the root to that node. If multiple nodes exist with label $V$, then one is chosen arbitrarily. If no nodes exist with label $V$, then the value false is returned (and, in particular, if the GetNode is located in a conditional, then the conditional fails).

The notation NewNode$(M_1 M_2 \cdots M_i U) \leftarrow Y$ is interpreted as follows for a sequence of $d$-bit strings $M_1, \ldots, M_i$ (where $i$ can be zero) and a $d$ bit string $U$ and an $n$-bit string $Y$. (1) Locate the node found by by following from the root the edges labeled by $M_1, M_2, \ldots, M_i$ (if $i = 0$, then this node is simply the root node). (2) Add an edge labeled $U$ from the found node to a new node with label $Y$. It is worth pointing out two facts. the path with edges labeled by $M_1, \ldots, M_i$ is guaranteed to exist since these labels are always returned by a call to GetNode. Second, each path's concatenation of edge labels is unique because we disallow pointless queries (particularly, no node will ever have two edges with the same label to two different children).

The right hand diagram in Figure 6 displays a sample tree after the queries $Y_1 = SB_f(0^d \,||\, IV)$, $Y_2 = SB_f(0^d \,||\, Y_1)$, $Y_3 = SB_f(1^d \,||\, IV)$, $Y_4 = SB_f(0^d \,||\, Y_3)$, and $Y_5 = SB_f(1^d \,||\, Y_3)$. Note that another query $SB_f(0^d \,||\, Y_6)$ for which $Y_6$ is not equal to any of the nodes in the tree will have no effect on the tree.

For queries to $SB_g$, the simulator checks if the first $n$ bits of the input are equal to the label of some node in the graph. If there is no such node, then a random $n$-bit string is returned. Otherwise, a sequence of edge labels are returned (corresponding to labels of the edges on the path from the root to the matching node). The simulator then queries its oracle $\mathcal{F}$ on the concatenation of these edge labels and the last $d$ bits of the input.

We now give some intuition regarding why the simulator can fool any distinguisher. The simulator's high-level goal is to behave like $f, g$ in a way consistent with $gf^+$. Towards this end it must return values that are consistent with the construction and two random oracles. Since the adversary never learns anything about $f$ range points except by querying the right oracle (even when dealing with the actual construction), the simulator can make up these values at random. Only when queries to $SB_g$ occur must the simulator check to ensure that values returned here correspond to values returned by $\mathcal{F}$. Intuitively, this is always possible as long as the adversary necessarily queries $SB_f$ for the intermediate values, and in order. In this case we are guaranteed to have a path in the tree corresponding to the message that the adversary is checking. Then, the only way to trick the simulator is for there to be a collision in intermediate values output or for the adversary to predict one of these intermediate values. As we show rigorously in the proof, neither can occur with high probability.

## 6.2 Bounding $A$'s Advantage

We utilize a game-playing argument. Thus, we replace the oracles by games that simulate them. We will notate this by $A^{\mathrm{G}i}$ where $\mathrm{G}i = \mathrm{G0}, \mathrm{G1}$, etc. Let $p_i = \Pr\left[\, A^{\mathrm{G}i} \Rightarrow 1 \,\right]$.

**(G0 and G1; Figure 7)** We start with a game G0 that simulates exactly the oracles $gf^+$, $f$, and $g$. The game, which includes the boxed statements, is shown in Figure 7. There are three interfaces, corresponding to the three types of oracle queries that can be made: $\mathsf{L}$, $\mathsf{R}_f$, and $\mathsf{R}_g$. Note that we increment the query identifier $t$ globally. We push the functionality of $\mathsf{L}$ into a subroutine $\mathsf{LSub}$, since this functionality is also sometimes utilized when answering $g$ right oracle queries (line 302). The tables $f$ and $g$ in the game track two lazily-sampled random functions, and the notation $Dom(f)$ is shorthand for the current set of all points defined on the domain of $f$. Although range points are chosen in $\mathsf{L}$ and $\mathsf{R}_f$ separately, the checks ensure consistency (lines 105, 109, 201, and 304). Clearly $\mathsf{L}$ is an exact simulation of $gf^+$. Although $\mathsf{R}_f$ tracks some of the input and returned values, this has no affect on the responses and so $\mathsf{R}_f$ implements a random oracle. Finally we argue that $\mathsf{R}_g$ also implements a random oracle. This is clear except in the case that line 302 is executed. But in this case, all the intermediate values $Y_i^t$ (for $1 \le i \le k-1$) used in $\mathsf{LSub}$ have already been defined by adversarial queries to $\mathsf{R}_f$. Thus all that possibly remains to do is sample a point for $Y_k^t$, which is the range point $g[X^t] = g[Y_{k-1}^t \,||\, M_k^t]$. Had there been an earlier query $s$ to $\mathsf{L}$ with message $M_1^t M_2^t \cdots M_i^t U^t$, then $\mathsf{LSub}$ simply returns the correct $g$ range point. Otherwise a new range point is chosen and returned. This behavior is consistent with implementing a random oracle $g$, we simply do it in a seemingly strange way.

Now we must justify that G1, which does not include the boxed statements, is a correct simulation of $\mathcal{F}$, $SB_f$, and $SB_g$. It is easy to verify that $\mathsf{L}$ returns random bits for any query. The game implements $\mathsf{R}_g$ identically to $SB_g$. Now to justify that $\mathsf{R}_f$ behaves like $SB_f$. If line 202 is never executed, then $\mathsf{R}_f$ calculates its responses exactly like $SB_f$. If line 202 is executed, then some

**Games G0 and G1**

Respond to the $t$-th query as follows:

A LEFT QUERY $\mathsf{L}(M^t)$:

$M_1^t \cdots M_k^t \xleftarrow{d} M^t$
**ret** $\mathsf{LSub}(t, M_1^t \cdots M_k^t)$

SUBROUTINE $\mathsf{LSub}(t, M_1^t \cdots M_k^t)$:

Let $s$ be min value s.t.
$\quad M_1^s \cdots M_k^s = M_1^t \cdots M_k^t$
**if** $s < t$ **then ret** $Y_k^s$
$Y_0 \leftarrow IV$
**for** $1 \le i \le k-1$
$\quad Y_i^t \xleftarrow{\$} \{0,1\}^n$
$\quad$ **if** $M_i^t \,||\, Y_{i-1}^t \in Dom(f)$ **then**
$\quad\quad Y_i^t \leftarrow f[M_i^t \,||\, Y_{i-1}^t]$
$\quad f[M_i^t \,||\, Y_{i-1}^t] \leftarrow Y_i^t$
$Y_k^t \xleftarrow{\$} \{0,1\}^n$
**if** $Y_{k-1}^t \,||\, M_k^t \in Dom(g)$ **then**
$\quad bad \leftarrow \mathsf{true}$
$\quad \boxed{Y_k^t \leftarrow g[Y_{k-1}^t \,||\, M_k^t]}$
$g[Y_{k-1}^t \,||\, M_k^t] \leftarrow Y_k^t$
**ret** $Y_k^t$

A RIGHT QUERY $\mathsf{R}_f(X^t)$:

$Y^t \xleftarrow{\$} \{0,1\}^n$
**if** $X^t \in Dom(f)$ **then**
$\quad Y^t \leftarrow f[X^t]$
Parse $X^t$ into $U^t \,||\, V^t$ s.t. $|U^t| = d$, $|V^t| = n$
**if** $V^t = IV$ **then**
$\quad \textsc{NewNode}(U^t) \leftarrow Y^t$
**if** $M_1^t M_2^t \cdots M_i^t \leftarrow \textsc{GetNode}(V^t)$ **then**
$\quad \textsc{NewNode}(M_1^t M_2^t \cdots M_i^t U^t) \leftarrow Y^t$
**ret** $f[X^t] \leftarrow Y^t$

A RIGHT QUERY $\mathsf{R}_g(X^t)$:

Parse $X^t$ into $V^t \,||\, U^t$ s.t. $|V^t| = n$, $|U^t| = d$
**if** $M_1^t M_2^t \cdots M_i^t \leftarrow \textsc{GetNode}(V^t)$ **then**
$\quad$ **ret** $\mathsf{LSub}(t, M_1^t M_2^t \cdots M_i^t U^t)$
$Y^t \xleftarrow{\$} \{0,1\}^n$
**if** $X^t \in Dom(g)$ **then**
$\quad bad \leftarrow \mathsf{true}$
$\quad \boxed{Y^t \leftarrow g[X^t]}$
**ret** $g[X^t] \leftarrow Y^t$

Figure 7: Games G0 (boxed statements included) and G1 (boxed statements removed).

**Game G2**

Respond to the $t$-th query as follows:

A LEFT QUERY $\mathsf{L}(M^t)$:

$M_1^t M_2^t \cdots M_k^t \xleftarrow{d} M^t$
**ret** $\mathsf{LSub}(t, M_1^t M_2^t \cdots M_k^t)$

SUBROUTINE $\mathsf{LSub}(t, M_1^t M_2^t \cdots M_k^t)$:

Let $s$ be min value s.t.
$\quad M_1^s M_2^s \cdots M_k^s = M_1^t M_2^t \cdots M_k^t$
**if** $s < t$ **then ret** $Y_k^s$
$Y_0 \leftarrow IV$
**for** $1 \le i \le k-1$
$\quad Y_i^t \xleftarrow{\$} \{0,1\}^n$
$\quad$ **if** $M_i^t \,||\, Y_{i-1}^t \in Dom(f)$ **then**
$\quad\quad Y_i^t \leftarrow f[M_i^t \,||\, Y_{i-1}^t]$
$\quad f[M_i^t \,||\, Y_{i-1}^t] \leftarrow Y_i^t$
**if** $Y_{k-1}^t \,||\, M_k^t \in Dom(g)$ **then** $bad \leftarrow \mathsf{true}$
**ret** $g[Y_{k-1}^t \,||\, M_k^t] \leftarrow Y_k^t \xleftarrow{\$} \{0,1\}^n$

A RIGHT QUERY $\mathsf{R}_f(X^t)$:

$Y^t \xleftarrow{\$} \{0,1\}^n$
**if** $X^t \in Dom(f)$ **then**
$\quad Y^t \leftarrow f[X^t]$
Parse $X^t$ into $U^t \,||\, V^t$ s.t. $|U^t| = d$, $|V^t| = n$
**if** $V^t = IV$ **then**
$\quad \textsc{NewNode}(U^t) \leftarrow Y^t$
**if** $M_1^t M_2^t \cdots M_i^t \leftarrow \textsc{GetNode}(V^t)$ **then**
$\quad \textsc{NewNode}(M_1^t M_2^t \cdots M_i^t U^t) \leftarrow Y^t$
**ret** $f[X^t] \leftarrow Y^t$

A RIGHT QUERY $\mathsf{R}_g(X^t)$:

Parse $X^t$ into $V^t \,||\, U^t$ s.t. $|V^t| = n$, $|U^t| = d$
**if** $M_1^t M_2^t \cdots M_i^t \leftarrow \textsc{GetNode}(V^t)$ **then**
$\quad$ **ret** $\mathsf{LSub}(t, M_1^t M_2^t \cdots M_i^t U^t)$
**if** $X^t \in Dom(g)$ **then** $bad \leftarrow \mathsf{true}$
**ret** $g[X^t] \leftarrow Y^t \xleftarrow{\$} \{0,1\}^n$

Figure 8: Game G2.

previous query to $\mathsf{L}$ defined $f[X^t]$ with a randomly selected value. The adversary learned nothing about this random choice since $\mathsf{L}$ always returns a string of random bits unrelated to the other random choices (recall that line 111 is removed from this game). Thus, although the random choice occurred previously, it is still 'fresh' and using it is equivalent to choosing a new random value. Since G0 and G1 are identical-until-bad we have that

$$p_0 - p_1 \leq \Pr\left[\, A^{\mathrm{G1}} \text{ sets } \mathsf{bad} \,\right].$$

The rest of this proof focuses on bounding this probability.

**(G1 → G2; Figure 8)** A conservative change consisting of delaying the random choice of $Y_k^t$ in $\mathsf{LSub}$ and $Y^t$ in $\mathsf{R}_g$ until after checking the corresponding domain points.

**(G2 → G3; Figure 9)** We replace our tracking of $g$ by a multiset $\mathcal{G}$ and defer the setting of *bad* until the finalization step. We must show that $\Pr\left[\, A^{\mathrm{G2}} \text{ sets } \mathsf{bad} \,\right] = \Pr\left[\, A^{\mathrm{G3}} \text{ sets } \mathsf{bad} \,\right]$. This must be true since G2 setting bad signifies the event that a domain point of $g$ is being redefined to some new random choice. Whenever this happens in G2, we are guaranteed to add a duplicate domain point to $\mathcal{G}$ in G3. Thus G3 will set bad whenever G2 would have.

**(G3 → G4; Figure 10)** We simply drop the random choices for $g$ range points. These have no impact on the ability of an adversary to set *bad*. More formally, we have that for any adversary $A$ we can create an adversary $B$ such that $\Pr\left[\, A^{\mathrm{G3}} \text{ sets } \mathsf{bad} \,\right] = \Pr\left[\, B^{\mathrm{G4}} \text{ sets } \mathsf{bad} \,\right]$. We define $B$ by simply modifying $A$ as follows: everywhere $A$ expects a response to a query to $\mathsf{L}$ or $\mathsf{R}_g$, we simply choose a random value and use this wherever the response was utilized in $A$.

**(G4 → G5; Figure 11)** A conservative change where we replace the call to $\mathsf{LSub}$ in $\mathsf{R}_g$ with code that implements the exact same behavior. In game G4, whenever $\mathsf{LSub}$ is called from $\mathsf{R}_g$, all that occurs is the possible addition of the value $X^t$ to $\mathcal{G}$. This is reflected by the new code in $\mathsf{R}_g$ in G5.

**(G5 → G6; Figure 12)** We push handling of queries to $\mathsf{L}$ to the finalization stage. To facilitate this, we introduce a query type variable $ty^t$ that is used to record which oracle was queried for query $t$. We must justify that it is conservative to defer the random choices made due to $\mathsf{L}$ queries until the finalization stage. Specifically we are discussing random choices for values $Y_i^t = f[M_i^t \,||\, Y_{i-1}^t]$. Recall that the adversary does not learn anything about a random choice of this form unless and until it queries $\mathsf{R}_f(M_i^t \,||\, Y_{i-1}^t)$. In the interactive portion of G6 these random choices are only made upon right oracle queries. During the finalization phase, any remaining random choices for values $Y_i^t$ are made. The distribution of these random variables remains unchanged, which in turn implies that the distribution of values added to $\mathcal{G}$ is equivalent. Thus $\Pr\left[\, B^{\mathrm{G5}} \text{ sets } \mathsf{bad} \,\right] = \Pr\left[\, B^{\mathrm{G6}} \text{ sets } \mathsf{bad} \,\right]$.

**(G6 → G7; Figure 13)** We move handling of $\mathsf{R}_f$ and $\mathsf{R}_g$ queries to the finalization stage. Note that these queries are still handled before queries to $\mathsf{L}$, which is identical to be the behavior of G6. Therefore $\Pr\left[\, B^{\mathrm{G6}} \text{ sets } \mathsf{bad} \,\right] = \Pr\left[\, B^{\mathrm{G7}} \text{ sets } \mathsf{bad} \,\right]$.

**(G7 → G8; Figure 14)** A lossy change in which G8 restricts sampling in $\mathsf{R}_f$ to not include collisions with 1) previously defined $f$ range points, 2) the last $n$ bits of previous $f$ domain points, and 3) the first $n$ bits of previous queries to $\mathsf{R}_g$. This is accomplished by maintaining another set $\mathcal{B}$ and adding the appropriate points to it. Note that we initially have $\mathcal{B} = \{IV\}$. Now to analyze the loss due to this restriction. For the $i^{th}$ query to $f$ we have $i-1$ domain and range points defined for $f$ and up to $q_g$ previous queries to $\mathsf{R}_g$ (and one point for $IV$). Thus

$$|\mathcal{B}| \leq 1 + 2(i-1) + q_g\,.$$

**Game G3**

Respond to the $t$-th query as follows:

A LEFT QUERY $\mathsf{L}(M^t)$:

$M_1^t M_2^t \cdots M_k^t \xleftarrow{d} M^t$
**ret** $\mathsf{LSub}(t, M_1^t M_2^t \cdots M_k^t)$

SUBROUTINE $\mathsf{LSub}(t, M^t = M_1^t M_2^t \cdots M_k^t)$:

Let $s$ be min value s.t.
$\quad M_1^s M_2^s \cdots M_k^s = M_1^t M_2^t \cdots M_k^t$
**if** $s < t$ **then ret** $Y_k^s$
$Y_0 \leftarrow IV$
**for** $1 \leq i \leq k-1$
$\quad Y_i^t \xleftarrow{\$} \{0,1\}^n$
$\quad$ **if** $M_i^t \,||\, Y_{i-1}^t \in Dom(f)$ **then**
$\quad\quad Y_i^t \leftarrow f[M_i^t \,||\, Y_{i-1}^t]$
$\quad f[M_i^t \,||\, Y_{i-1}^t] \leftarrow Y_i^t$
$\mathcal{G} \xleftarrow{\cup} Y_{k-1}^t \,||\, M_k^t$
**ret** $Y_k^t \xleftarrow{\$} \{0,1\}^n$

Finalization:

$bad \leftarrow \exists X, X' \in \mathcal{G}$ s.t. $X = X'$

A RIGHT QUERY $\mathsf{R}_f(X^t)$:

$Y^t \xleftarrow{\$} \{0,1\}^n$
**if** $X^t \in Dom(f)$ **then**
$\quad Y^t \leftarrow f[X^t]$
Parse $X^t$ into $U^t \,||\, V^t$ s.t. $|U^t| = d$, $|V^t| = n$
**if** $V^t = IV$ **then**
$\quad$ NEWNODE$(U^t) \leftarrow Y^t$
**if** $M_1^t M_2^t \cdots M_i^t \leftarrow$ GETNODE$(V^t)$ **then**
$\quad$ NEWNODE$(M_1^t M_2^t \cdots M_i^t U^t) \leftarrow Y^t$
**ret** $f[X^t] \leftarrow Y^t$

A RIGHT QUERY $\mathsf{R}_g(X^t)$:

Parse $X^t$ into $V^t \,||\, U^t$ s.t. $|V^t| = n$, $|U^t| = d$
**if** $M_1^t M_2^t \cdots M_i^t \leftarrow$ GETNODE$(V^t)$ **then**
$\quad$ **ret** $\mathsf{LSub}(t, M_1^t M_2^t \cdots M_i^t U^t)$
$\mathcal{G} \xleftarrow{\cup} X^t$
**ret** $Y^t \xleftarrow{\$} \{0,1\}^n$

Figure 9: Game G3.

**Game G4**

Respond to the $t$-th query as follows:

A LEFT QUERY $\mathsf{L}(M^t)$:

$M_1^t M_2^t \cdots M_k^t \xleftarrow{d} M^t$
**ret** $\mathsf{LSub}(t, M_1^t M_2^t \cdots M_k^t)$

SUBROUTINE $\mathsf{LSub}(t, M_1^t M_2^t \cdots M_k^t)$:

Let $s$ be min value s.t.
$\quad M_1^s M_2^s \cdots M_k^s = M_1^t M_2^t \cdots M_k^t$
**if** $s < t$ **then ret** $Y_k^s$
$Y_0 \leftarrow IV$
**for** $1 \leq i \leq k-1$
$\quad Y_i^t \xleftarrow{\$} \{0,1\}^n$
$\quad$ **if** $M_i^t \,||\, Y_{i-1}^t \in Dom(f)$ **then**
$\quad\quad Y_i^t \leftarrow f[M_i^t \,||\, Y_{i-1}^t]$
$\quad f[M_i^t \,||\, Y_{i-1}^t] \leftarrow Y_i^t$
$\mathcal{G} \xleftarrow{\cup} Y_{k-1}^t \,||\, M_k^t$

Finalization:

$bad \leftarrow \exists X, X' \in \mathcal{G}$ s.t. $X = X'$

A RIGHT QUERY $\mathsf{R}_f(X^t)$:

$Y^t \xleftarrow{\$} \{0,1\}^n$
**if** $X^t \in Dom(f)$ **then**
$\quad Y^t \leftarrow f[X^t]$
Parse $X^t$ into $U^t \,||\, V^t$ s.t. $|U^t| = d$, $|V^t| = n$
**if** $V^t = IV$ **then**
$\quad$ NEWNODE$(U^t) \leftarrow Y^t$
**if** $M_1^t M_2^t \cdots M_i^t \leftarrow$ GETNODE$(V^t)$ **then**
$\quad$ NEWNODE$(M_1^t M_2^t \cdots M_i^t U^t) \leftarrow Y^t$
**ret** $f[X^t] \leftarrow Y^t$

A RIGHT QUERY $\mathsf{R}_g(X^t)$:

Parse $X^t$ into $V^t \,||\, U^t$ s.t. $|V^t| = n$, $|U^t| = d$
**if** $M_1^t M_2^t \cdots M_i^t \leftarrow$ GETNODE$(V^t)$ **then**
$\quad \mathsf{LSub}(t, M_1^t M_2^t \cdots M_i^t U^t)$
**else**
$\quad \mathcal{G} \xleftarrow{\cup} X^t$

Figure 10: Game G4.

**Game G5**

Respond to the $t$-th query as follows:

<u>A LEFT QUERY $\mathsf{L}(M^t)$:</u>

$M_1^t M_2^t \cdots M_k^t \xleftarrow{d} M^t$
**ret** $\mathsf{LSub}(t, M_1^t M_2^t \cdots M_k^t)$

<u>SUBROUTINE $\mathsf{LSub}(t, M_1^t M_2^t \cdots M_k^t)$:</u>

Let $s$ be min value s.t.
  $M_1^s M_2^s \cdots M_k^s = M_1^t M_2^t \cdots M_k^t$
**if** $s = t$ **then**
  $Y_0 \leftarrow IV$
  **for** $1 \le i \le k-1$
    $Y_i^t \xleftarrow{\$} \{0,1\}^n$
    **if** $M_i^t \,||\, Y_{i-1}^t \in Dom(f)$ **then**
      $Y_i^t \leftarrow f[M_i^t \,||\, Y_{i-1}^t]$
    $f[M_i^t \,||\, Y_{i-1}^t] \leftarrow Y_i^t$
  $\mathcal{G} \xleftarrow{\cup} Y_{k-1}^t \,||\, M_k^t$

<u>Finalization:</u>

$bad \leftarrow \exists X, X' \in \mathcal{G}$ s.t. $X = X'$

<u>A RIGHT QUERY $\mathsf{R}_f(X^t)$:</u>

$Y^t \xleftarrow{\$} \{0,1\}^n$
**if** $X^t \in Dom(f)$ **then**
  $Y^t \leftarrow f[X^t]$
Parse $X^t$ into $U^t \,||\, V^t$ s.t. $|U^t| = d$, $|V^t| = n$
**if** $V^t = IV$ **then**
  NEWNODE$(U^t) \leftarrow Y^t$
**if** $M_1^t M_2^t \cdots M_i^t \leftarrow$ GETNODE$(V^t)$ **then**
  NEWNODE$(M_1^t M_2^t \cdots M_i^t U^t) \leftarrow Y^t$
**ret** $f[X^t] \leftarrow Y^t$

<u>A RIGHT QUERY $\mathsf{R}_g(X^t)$:</u>

Parse $X^t$ into $V^t \,||\, U^t$ s.t. $|V^t| = n$, $|U^t| = d$
**if** $M_1^t M_2^t \cdots M_i^t \leftarrow$ GETNODE$(V^t)$ **then**
  Let $s$ be smallest index s.t.
    $M_1^t M_2^t \cdots M_k^t = M_1^t M_2^t \cdots M_i^t U^t$
  **if** $s = t$ **then** $\mathcal{G} \xleftarrow{\cup} X^t$
**else**
  $\mathcal{G} \xleftarrow{\cup} X^t$

Figure 11: Game G5.

**Game G6**

Respond to the $t$-th query as follows:

<u>A LEFT QUERY $\mathsf{L}(M^t)$:</u>

$ty^t \leftarrow$ LEFT

<u>SUBROUTINE $\mathsf{LSub}(t, M_1^t M_2^t \cdots M_k^t)$:</u>

Let $s$ be min value s.t.
  $M_1^s M_2^s \cdots M_k^s = M_1^t M_2^t \cdots M_k^t$
**if** $s = t$ **then**
  $Y_0 \leftarrow IV$
  **for** $1 \le i \le k-1$
    $Y_i^t \xleftarrow{\$} \{0,1\}^n$
    **if** $M_i^t \,||\, Y_{i-1}^t \in Dom(f)$ **then**
      $Y_i^t \leftarrow f[M_i^t \,||\, Y_{i-1}^t]$
    $f[M_i^t \,||\, Y_{i-1}^t] \leftarrow Y_i^t$
  $\mathcal{G} \xleftarrow{\cup} Y_{k-1}^t \,||\, M_k^t$

<u>Finalization:</u>

**for** $1 \le j \le q$:
  **if** $ty^j =$ LEFT **then**
    $\mathsf{LSub}(M^j)$
$bad \leftarrow \exists X, X' \in \mathcal{G}$ s.t. $X = X'$

<u>A RIGHT QUERY $\mathsf{R}_f(X^t)$:</u>

$ty^t \leftarrow$ RIGHT-F
$Y^t \xleftarrow{\$} \{0,1\}^n$
Parse $X^t$ into $U^t \,||\, V^t$ s.t. $|U^t| = d$, $|V^t| = n$
**if** $V^t = IV$ **then**
  NEWNODE$(U^t) \leftarrow Y^t$
**if** $M_1^t M_2^t \cdots M_k^t \leftarrow$ GETNODE$(V^t)$ **then**
  NEWNODE$(M_1^t M_2^t \cdots M_k^t U^t) \leftarrow Y^t$
**ret** $f[X^t] \leftarrow Y^t$

<u>A RIGHT QUERY $\mathsf{R}_g(X^t)$:</u>

$ty^t \leftarrow$ RIGHT-G
Parse $X^t$ into $V^t \,||\, U^t$ s.t. $|V^t| = n$, $|U^t| = d$
**if** $M_1^t M_2^t \cdots M_i^t \leftarrow$ GETNODE$(V^t)$ **then**
  Let $s$ be smallest index s.t.
    $M_1^t M_2^t \cdots M_k^t = M_1^t M_2^t \cdots M_i^t U^t$
  **if** $s = t$ **then** $\mathcal{G} \xleftarrow{\cup} X^t$
**else**
  $\mathcal{G} \xleftarrow{\cup} X^t$

Figure 12: Game G6.

**Game G7**

Respond to $t$-th query as follows:

A LEFT QUERY $\mathsf{L}(M^t)$:

$ty^t \leftarrow$ LEFT

SUBROUTINE $\mathsf{LSub}(t, M_1^t M_2^t \cdots M_k^t)$:

Let $s$ be min value s.t.
$\quad M_1^s M_2^s \cdots M_k^s = M_1^t M_2^t \cdots M_k^t$
**if** $s = t$ **then**
$\quad Y_0 \leftarrow IV$
$\quad$**for** $1 \leq i \leq k-1$
$\quad\quad Y_i^t \xleftarrow{\$} \{0,1\}^n$
$\quad\quad$**if** $M_i^t \mid\mid Y_{i-1}^t \in Dom(f)$ **then**
$\quad\quad\quad Y_i^t \leftarrow f[M_i^t \mid\mid Y_{i-1}^t]$
$\quad\quad f[M_i^t \mid\mid Y_{i-1}^t] \leftarrow Y_i^t$
$\quad \mathcal{G} \xleftarrow{\cup} Y_{k-1}^t \mid\mid M_k^t$

Finalization:

**for** $1 \leq j \leq q$:
$\quad$**if** $ty^j =$ RIGHT-F **then**
$\quad\quad \mathsf{RSub}_f(j, X^j)$
$\quad$**if** $ty^j =$ RIGHT-G **then**
$\quad\quad \mathsf{RSub}_g(j, X^j)$
**for** $1 \leq j \leq q$:
$\quad$**if** $ty^j =$ LEFT **then**
$\quad\quad \mathsf{LSub}(M^j)$
$bad \leftarrow \exists X, X' \in \mathcal{G}$ s.t. $X = X'$

A RIGHT QUERY $\mathsf{R}_f(X^t)$:

$ty^t \leftarrow$ RIGHT-F
**ret** $Y^t \xleftarrow{\$} \{0,1\}^n$

A RIGHT QUERY $\mathsf{R}_g(X^t)$:

$ty^t \leftarrow$ RIGHT-G

SUBROUTINE $\mathsf{RSub}_f(t, X^t)$:

Parse $X^t$ into $U^t \mid\mid V^t$ s.t. $|U^t| = d$, $|V^t| = n$
**if** $V^t = IV$ **then**
$\quad$NEWNODE$(U^t) \leftarrow Y^t$
**if** $M_1^t M_2^t \cdots M_i^t \leftarrow$ GETNODE$(V^t)$ **then**
$\quad$NEWNODE$(M_1^t M_2^t \cdots M_i^t U^t) \leftarrow Y^t$
$f[X^t] \leftarrow Y^t$

SUBROUTINE $\mathsf{RSub}_g(t, X^t)$:

Parse $X^t$ into $V^t \mid\mid U^t$ s.t. $|V^t| = n$, $|U^t| = d$
**if** $M_1^t M_2^t \cdots M_i^t \leftarrow$ GETNODE$(V^t)$ **then**
$\quad$Let $s$ be smallest index s.t.
$\quad\quad M_1^s M_2^s \cdots M_k^s = M_1^t M_2^t \cdots M_i^t U^t$
$\quad$**if** $s = t$ **then** $\mathcal{G} \xleftarrow{\cup} X^t$
**else**
$\quad \mathcal{G} \xleftarrow{\cup} X^t$

Figure 13: Game G7.

The probability then that none of these points would be selected by a random selection from $\{0,1\}^n$ is

$$\Pr\left[Y \xleftarrow{\$} \{0,1\}^n : Y \in \mathcal{B}\right] \leq \frac{2i - 1 + q_g}{2^n}.$$

Now we want to sum up this probability over all queries to $\mathsf{R}_f$ (i.e., apply the union bound):

$$\begin{aligned}\sum_{i=1}^{q_f} \frac{2i-1+q_g}{2^n} &= \frac{1}{2^n}\left(\sum_{i=1}^{q_f} 2i - \sum_{i=1}^{q_f} 1\right) + \frac{q_g}{2^n}\sum_{i=1}^{q_f} 1 \\ &= \frac{q_f^2 + q_g q_f}{2^n}.\end{aligned}$$

Therefore we have that

$$\Pr\left[B^{\mathrm{G7}} \text{ sets } \mathsf{bad}\right] \leq \Pr\left[B^{\mathrm{G8}} \text{ sets } \mathsf{bad}\right] + \frac{q_f^2 + q_g q_f}{2^n}.$$

Note that this change does not affect the random choices made in $\mathsf{LSub}$ in the finalization stage — these might still cause collisions.

**(G8 → G9; Figure 15)** The last game is non-interactive. The adversary specifies a fixed transcript that maximizes the probability of setting the flag *bad*. A transcript is

$$\tau = (ty^1, \mathtt{S}^1, \mathtt{Y}^1), (ty^2, \mathtt{S}^2, \mathtt{Y}^2), \ldots, (ty^q, \mathtt{S}^q, \mathtt{Y}^q)$$

**Game G8**

Respond to $t$-th query as follows:

A LEFT QUERY $\mathsf{L}(M^t)$:

000 $ty^t \leftarrow$ LEFT

SUBROUTINE $\mathsf{LSub}(t, M_1^t M_2^t \cdots M_k^t)$:

100 Let $s$ be min value s.t. $M_1^s M_2^s \cdots M_k^s = M_1^t M_2^t \cdots M_k^t$
101 **if** $s = t$ **then**
102 $Y_0 \leftarrow IV$
103 **for** $1 \le i \le k-1$
104 $Y_i^t \xleftarrow{\$} \{0,1\}^n$
105 **if** $M_i^t \mathbin{||} Y_{i-1}^t \in Dom(f)$ **then**
106 $Y_i^t \leftarrow f[M_i^t \mathbin{||} Y_{i-1}^t]$
107 $f[M_i^t \mathbin{||} Y_{i-1}^t] \leftarrow Y_i^t$
108 $\mathcal{G} \xleftarrow{\cup} Y_{k-1}^t \mathbin{||} M_k^t$

Finalization:

400 **for** $1 \le j \le q$:
401 **if** $ty^j =$ RIGHT-F **then**
402 $\mathsf{RSub}_f(j, X^j)$
403 **if** $ty^j =$ RIGHT-G **then**
404 $\mathsf{RSub}_g(j, X^j)$
405 **for** $1 \le j \le q$:
406 **if** $ty^j =$ LEFT **then**
407 $\mathsf{LSub}(M^j)$
408 $bad \leftarrow \exists X, X' \in \mathcal{G}$ s.t. $X = X'$

A RIGHT QUERY $\mathsf{R}_f(X^t)$:

200 $ty^t \leftarrow$ RIGHT-F
201 $Y^t \xleftarrow{\$} \{0,1\}^n / \mathcal{B}$
202 $\mathcal{B} \xleftarrow{\cup}$ last $n$ bits of $X^t$
203 $\mathcal{B} \xleftarrow{\cup} Y^t$
204 **ret** $Y^t$

A RIGHT QUERY $\mathsf{R}_g(X^t)$:

300 $ty^t \leftarrow$ RIGHT-G
301 $\mathcal{B} \xleftarrow{\cup}$ first $n$ bits of $X^t$

SUBROUTINE $\mathsf{RSub}_f(t, X^t)$:

500 Parse $X^t$ into $U^t \mathbin{||} V^t$ s.t. $|U^t| = d$, $|V^t| = n$
501 **if** $V^t = IV$ **then**
502 $\textsc{NewNode}(U^t) \leftarrow Y^t$
503 **if** $M_1^t M_2^t \cdots M_i^t \leftarrow \textsc{GetNode}(V^t)$ **then**
504 $\textsc{NewNode}(M_1^t M_2^t \cdots M_i^t U^t) \leftarrow Y^t$
505 $f[X^t] \leftarrow Y^t$

SUBROUTINE $\mathsf{RSub}_g(t, X^t)$:

600 Parse $X^t$ into $V^t \mathbin{||} U^t$ s.t. $|V^t| = n$, $|U^t| = d$
601 **if** $M_1^t M_2^t \cdots M_i^t \leftarrow \textsc{GetNode}(V^t)$ **then**
602 Let $s$ be smallest index s.t. $M_1^s M_2^s \cdots M_k^s = M_1^t M_2^t \cdots M_i^t U^t$
603 **if** $s = t$ **then** $\mathcal{G} \xleftarrow{\cup} X^t$
604 **else**
605 $\mathcal{G} \xleftarrow{\cup} X^t$

Figure 14: Game G8. Initially $\mathcal{B} = \{IV\}$.

where $\mathtt{ty}^i \in \{\textsc{Left}, \textsc{Right-f}, \textsc{Right-g}\}$; and $\mathtt{S}^i \in \{0,1\}^{d+n}$ if $\mathtt{ty}^i \neq \textsc{Left}$ (we'll notate these inputs by $\mathtt{X}^i$) or $\mathtt{S}^i \in (\{0,1\}^n)^+$ if $ty^i = \textsc{Left}$ (notated by $\mathtt{M}^i$); and $\mathtt{Y}^i \in \{0,1\}^n$ if $\mathtt{ty}^i = \textsc{Right-f}$ or is otherwise $\epsilon$. We additionally constrain tuples $(\textsc{Right-f}^j, \mathtt{S}^j, \mathtt{Y}^j)$. We have that $\mathtt{Y}^j$ can never be specified to be $IV$. Additionally, for $i < j$ we have the following three constraints: 1) $\mathtt{Y}^j \neq \mathtt{Y}^i$ for tuple $(\textsc{Right-f}^i, \mathtt{S}^i, \mathtt{Y}^i)$; 2) $\mathtt{Y}^j$ can not equal the last $n$ bits of $\mathtt{S}^i$ for a tuple $(\textsc{Right-f}^i, \mathtt{S}^i, \mathtt{Y}^i)$; and 3) $\mathtt{Y}^j$ can not equal the first $n$ bits of $\mathtt{S}^i$ for a tuple $(\textsc{Right-g}^i, \mathtt{S}^i, \epsilon)$. We have that $\Pr\left[\, B^{\mathrm{G8}} \text{ sets } \mathsf{bad} \,\right] \le \Pr\left[\, B^{\mathrm{G9}} \text{ sets } \mathsf{bad} \,\right]$.

**(Analysis of G9)** We now bound the probability of *bad* being set in game G9 by investigating all potential collisions in $\mathcal{G}$. We break our analysis into cases based on how elements are added to $\mathcal{G}$.

First consider two values added to $\mathcal{G}$ via line 108. Let the transcript entries responsible for the two values be $(\textsc{Left}, \mathtt{M}^s, \epsilon)$ and $(\textsc{Left}, \mathtt{M}^t, \epsilon)$ where $\mathtt{M}^s = \mathtt{M}_1^s \cdots \mathtt{M}_k^s$ and $\mathtt{M}^t = \mathtt{M}_1^t \cdots \mathtt{M}_c^t$ where $c$ and $k$ are not necessarily equal. Then for a collision to occur we know that $Y_{k-1}^s = Y_{c-1}^t$. If either of these values was picked in the finalization phase (not specified by the transcript), then clearly the probability of them being equal is $2^{-n}$. Otherwise they were both specified by transcript entries $(\textsc{Right-f}, \mathtt{X}^i, \mathtt{Y}^i)$ where $Y_{k-1}^s = \mathtt{Y}^i$ and $(\textsc{Right-f}, \mathtt{X}^j, \mathtt{Y}^j)$ where $Y_{c-1}^t = \mathtt{Y}^j$. By our restrictions on the transcript we then know that $\mathtt{X}^i = \mathtt{X}^j$, or rather that $\mathtt{M}_{k-2}^s \mathbin{||} Y_{k-2}^s = \mathtt{M}_{c-2}^t \mathbin{||} Y_{c-2}^t$. We then repeat the above reasoning, finishing if ever a fresh choice is made in the finalization phase (with probability of collision being $2^{-n}$). We will show that such a fresh choice must be made. Suppose otherwise. If $s = t$ (the messages are the same number of blocks) then by the above reasoning $\texttt{M}^t = \texttt{M}^s$, which is not allowed in the transcript (no pointless queries). If (wlog) $s < t$, then we have that the adversary necessarily specified a transcript entry $(\textsc{Right-f}, \texttt{X}, IV)$ which is not allowed.

Now consider a value $Y^s_{k-1} \,||\, \texttt{M}^s_k$ added to $\mathcal{G}$ via line 108 for a transcript entry $(\textsc{Left}, \texttt{M}^s, \epsilon)$ and a value $\texttt{X}^t$ added via line 603 for a transcript entry $(\textsc{Right-g}, \texttt{M}^t, \epsilon)$. This is just a special case of above (for two values added via line 108). We simply iterate backwards over the blocks of $s$, knowing that at each the corresponding blocks for $t$ are specified in the transcript. The same reasoning gives us that the probability of a collision is $2^{-n}$.

Finally consider a value added to $\mathcal{G}$ via line 108 for a transcript entry $(\textsc{Left}, \texttt{M}^s, \epsilon)$ and a value added via line 605 for a transcript entry $(\textsc{Right-g}, \texttt{X}^t, \epsilon)$. We will reason about the choices of $Y^s_i$ values in $\mathsf{LSub}$ when $\texttt{M}^s$ is handled, starting with $i = k - 1$ and working backwards. There are two cases to consider for $Y^s_{k-1}$. If $Y^s_{k-1}$ is a fresh random choice then we are done, as it will collide with the first $n$ bits of $\texttt{X}^t$ with probability $2^{-n}$. Suppose then that $Y^s_{k-1} = \texttt{Y}^j$ for an entry $(\textsc{Right-f}, \texttt{X}^j, \texttt{Y}^j)$, where the first $d$ bits of $\texttt{S}^j$ equal $\texttt{M}^s_{k-2}$. Necessarily $j < t$, since otherwise the transcript would not be allowed. Now we switch to looking at $Y^s_{k-2}$ and analyze the probability that it is equal to the last $n$ bits of $\texttt{S}^j$. We apply similar reasoning: if it is a fresh random choice then the probability of collision is $2^{-n}$, otherwise an entry $(\textsc{Right-f}, \texttt{S}^l, \texttt{Y}^l)$ exists in the transcript with $Y^s_{k-2} = \texttt{Y}^l$ and with $l < j$. Eventually we are guaranteed that some value $Y^s_i$ must be a fresh random choice, otherwise all of these values would have been specified by $\textsc{Right-f}$ entries that are located in the transcript before $t$. If that were the case then our tree structure would contain a node labeled by the first $n$ bits of $\texttt{X}^t$ and the path to the node would be labeled by $\texttt{M}^s_1, \ldots, \texttt{M}^s_k = \texttt{M}^s$ resulting in line 605 never being executed. Thus a collision of this kind can occur with probability no greater than $2^{-n}$.

In all cases the probability of a collision between a pair of points in $\mathcal{G}$ is $2^{-n}$. Since the multiset $\mathcal{G}$ has at most $q_L + q_g$ elements, we have that

$$
\begin{aligned}
\Pr\left[\, B^{\mathrm{G9}} \text{ sets } \mathsf{bad} \,\right] &\leq \binom{q_L + q_g}{2} \frac{1}{2^n} \\
&= \frac{0.5(q_L + q_g)(q_l + q_g - 1)}{2^n} \\
&= \frac{0.5(q_L + q_g)^2 - 0.5(q_L + q_g)}{2^n} .
\end{aligned}
$$

Collecting the above game transitions, we can bound $p$:

$$
\begin{aligned}
p &\leq \Pr\left[\, A^{\mathrm{G0}} \text{ sets } \mathsf{bad} \,\right] = \Pr\left[\, A^{\mathrm{G3}} \text{ sets } \mathsf{bad} \,\right] = \Pr\left[\, A^{\mathrm{G4}} \text{ sets } \mathsf{bad} \,\right] \\
&= \Pr\left[\, B^{\mathrm{G4}} \text{ sets } \mathsf{bad} \,\right] = \Pr\left[\, B^{\mathrm{G5}} \text{ sets } \mathsf{bad} \,\right] = \Pr\left[\, B^{\mathrm{G6}} \text{ sets } \mathsf{bad} \,\right] \\
&= \Pr\left[\, B^{\mathrm{G7}} \text{ sets } \mathsf{bad} \,\right] \leq \Pr\left[\, B^{\mathrm{G8}} \text{ sets } \mathsf{bad} \,\right] + \frac{q_f^2 + q_g q_f}{2^n} \\
&= \Pr\left[\, B^{\mathrm{G9}} \text{ sets } \mathsf{bad} \,\right] + \frac{q_f^2 + q_g q_f}{2^n} \\
&\leq \frac{0.5(q_L + q_g)^2 - 0.5(q_L + q_g)}{2^n} + \frac{q_f^2 + q_g q_f}{2^n} \\
&\leq \frac{(q_L + q_g)^2 + q_f^2 + q_g q_f}{2^n} .
\end{aligned}
$$

**Game G9**

Respond to $t$-th query as follows:

Given transcript $\tau = (ty^1, \mathtt{S}^i, \mathtt{Y}^i), \ldots, (ty^q, \mathtt{S}^q, \mathtt{Y}^q)$:

**for** $1 \le j \le q$:
  **if** $ty^j =$ RIGHT-F **then**
    $\mathsf{RSub}_f(j, \mathtt{S}^j, \mathtt{Y}^j)$
  **if** $ty^j =$ RIGHT-G **then**
    $\mathsf{RSub}_g(j, \mathtt{S}^j)$
**for** $1 \le j \le q$:
  **if** $ty^j =$ LEFT **then**
    $\mathsf{LSub}(j, \mathtt{S}^j)$
$bad \leftarrow \exists X, X' \in \mathcal{G}$ s.t. $X = X'$

SUBROUTINE $\mathsf{LSub}(t, \mathtt{M}_1^t \mathtt{M}_2^t \cdots \mathtt{M}_k^t)$:

Let $s$ be min value s.t.
  $\mathtt{M}_1^s \mathtt{M}_2^s \cdots \mathtt{M}_k^s = \mathtt{M}_1^t \mathtt{M}_2^t \cdots \mathtt{M}_k^t$
**if** $s = t$ **then**
  $Y_0 \leftarrow IV$
  **for** $1 \le i \le k-1$
    $Y_i^t \xleftarrow{\$} \{0,1\}^n$
    **if** $\mathtt{M}_i^t \mid\mid Y_{i-1}^t \in Dom(f)$ **then**
      $Y_i^t \leftarrow f[\mathtt{M}_i^t \mid\mid Y_{i-1}^t]$
    $f[\mathtt{M}_i^t \mid\mid Y_{i-1}^t] \leftarrow Y_i^t$
  $\mathcal{G} \xleftarrow{\cup} Y_{k-1}^t \mid\mid \mathtt{M}_k^t$

SUBROUTINE $\mathsf{RSub}_f(t, \mathtt{X}^t, \mathtt{Y}^t)$:

Parse $\mathtt{X}^t$ into $\mathtt{U}^t \mid\mid \mathtt{V}^t$ s.t. $|\mathtt{U}^t| = d$, $|\mathtt{V}^t| = n$
**if** $\mathtt{V}^t = IV$ **then**
  NEWNODE$(\mathtt{U}^t) \leftarrow \mathtt{Y}^t$
**if** $\mathtt{M}_1^t \mathtt{M}_2^t \cdots \mathtt{M}_i^t \leftarrow$ GETNODE$(V^t)$ **then**
  NEWNODE$(\mathtt{M}_1^t \mathtt{M}_2^t \cdots \mathtt{M}_i^t \mathtt{U}^t) \leftarrow \mathtt{Y}^t$
$f[\mathtt{X}^t] \leftarrow \mathtt{Y}^t$

SUBROUTINE $\mathsf{RSub}_g(t, \mathtt{X}^t)$:

Parse $\mathtt{X}^t$ into $\mathtt{V}^t \mid\mid \mathtt{U}^t$ s.t. $|\mathtt{V}^t| = n$, $|\mathtt{U}^t| = d$
**if** $\mathtt{M}_1^t \mathtt{M}_2^t \cdots \mathtt{M}_i^t \leftarrow$ GETNODE$(\mathtt{V}^t)$ **then**
  Let $s$ be smallest index s.t.
    $\mathtt{M}_1^s \mathtt{M}_2^s \cdots \mathtt{M}_k^s = \mathtt{M}_1^t \mathtt{M}_2^t \cdots \mathtt{M}_i^t \mathtt{U}^t$
  **if** $s = t$ **then** $\mathcal{G} \xleftarrow{\cup} \mathtt{X}^t$
**else**
  $\mathcal{G} \xleftarrow{\cup} \mathtt{X}^t$

Figure 15: Game G9.


## Acknowledgments

We would like to thank Thomas Shrimpton for valuable feedback on a draft of this paper.


## References


[1] An, J.H., Bellare, M.: Constructing VIL-MACs from FIL-MACs: message authentication under weakened assumptions. In: Advances in Cryptology - CRYPTO '99. Volume 1666 of Lecture Notes in Computer Science, Springer (1999) 252–269.

[2] Bellare, M.: New Proofs for NMAC and HMAC: Security Without Collision-Resistance. In: Advances in Cryptology - CRYPTO '06. Volume 4117 of Lecture Notes in Computer Science, Springer (2006) 602–619.

[3] Bellare, M., Boldyreva, A., Palacio, A.: An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem. In Cachin, C., Camenisch, J., eds.: Advances in Cryptology - EUROCRYPT '04. Volume 3027 of Lecture Notes in Computer Science, Springer (2004) 171–188.

[4] Bellare, M., Canetti, R., Krawczyk, H.: Keying Hash Functions for Message Authentication. In: Advances in Cryptology - CRYPTO '96. Volume 1109 of Lecture Notes in Computer Science, Springer (1996) 1–15.

[5] Bellare, M., Canetti, R., Krawczyk, H.: Pseudorandom functions revisited: the cascade construction and its concrete security. In: FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science, IEEE Computer Society (1996) 514–523.

[6] Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: CCS '93, ACM Press (1993) 62–73.

[7] Bellare, M., Rogaway, P.: Optimal Asymmetric Encryption. In: Advances in Cryptology - EUROCRYPT '94. Volume 950 of Lecture Notes in Computer Science, Springer (1994) 92–111.

[8] Bellare, M., Rogaway, P.: The Exact Security of Digital Signatures - How to Sign with RSA and Rabin. In: Advances in Cryptology - EUROCRYPT '96. Volume 1070 of Lecture Notes in Computer Science, Springer (1996) 399–416.

[9] Bellare, M., Rogaway, P.: Collision-Resistant Hashing: Towards Making UOWHFs Practical. In: Advances in Cryptology - CRYPTO '97. Volume 1294 of Lecture Notes in Computer Science, Springer (1997) 470–484.

[10] Bellare, M., Rogaway, P.: The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In: Advances in Cryptology - EUROCRYPT '06. Volume 4004 of Lecture Notes in Computer Science, Springer (2006) 409–426.

[11] Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. J. ACM **51**(4) (2004) 557–594.

[12] Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgard Revisited: How to Construct a Hash Function. In: Advances in Cryptology - CRYPTO '05. Volume 3621 of Lecture Notes in Computer Science, Springer (2004) 21–39.

[13] Damgård, I.: A design principle for hash functions. In: Advances in Cryptology - CRYPTO '89. Volume 435 of Lecture Notes in Computer Science, Springer (1989) 416–427.

[14] Maurer, U.M., Renner, R., Holenstein, C.: Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In: TCC '04. Volume 2951 of Lecture Notes in Computer Science, Springer (2004) 21–39.

[15] Maurer, U., Sjödin, J.: Single-key AIL-MACs from any FIL-MAC. In: ICALP '05. Volume 3580 of Lecture Notes in Computer Science, Springer (2005) 472–484.

[16] Merkle, R.C.: One way hash functions and DES. In: Advances in Cryptology - CRYPTO '89. Volume 435 of Lecture Notes in Computer Science, Springer (1989) 428–446.

[17] National Institute of Standards and Technology: FIPS PUB 180-1: Secure Hash Standard. (1995) Supersedes FIPS PUB 180 1993 May 11.

[18] RSA Laboratories: RSA PKCS #1 v2.1: RSA Cryptography Standards (2002).

[19] Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Advances in Cryptology - CRYPTO '05. Volume 3621 of Lecture Notes in Computer Science, Springer (2005) 17–36.

[20] Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Advances in Cryptology - EUROCRYPT '05. Volume 3494 of Lecture Notes in Computer Science, Springer (2005) 19–35.


# A Families of Compression and Hash Functions

Practical cryptographic compression and hash functions do not come equipped with a dedicated key input. As the signature $h\colon \{0,1\}^{d+n} \to \{0,1\}^n$ of a compression function indicates, it takes a single input. (Which we view as the concatenation of a $d$-bit data block with a $n$-bit chaining variable.) The hash function $H^h\colon \{0,1\}^* \to \{0,1\}^n$ correspondingly takes a single data input and returns an $n$-bit output. In contrast, a primitive like a block cipher has an explicit, dedicated key input and defines a family of functions, one per key. The absence of such an input is why using compression or hash functions to build PRFs or MACs requires "keying" [4]. One typically keys via the chaining variable.

One can, however, consider designing compression and hash functions which have a dedicated key input. In that case, the signature of the compression function becomes $h\colon \{0,1\}^k \times \{0,1\}^{d+n} \to \{0,1\}^n$ while that of the hash function is $H^h\colon \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^n$. We now have families of functions in the sense that for each key $K \in \{0,1\}^k$ we have functions $h(K,\cdot)$ and $H^h(K,\cdot)$ with the old signatures. For collision-resistance, a key $K$ is chosen at random and made public, but when we want to use the compression or hash functions as MACs or PRFs, we can use a secret key $K$.

This setting is adopted in many works [1, 9, 15]. In particular, in this setting, An and Bellare [1] initiated the consideration of MAC preservation and provided a MAC preserving transform. Maurer and Sjödin later proposed the Chain-Shift construction [15]. The Chain-Shift and EMD constructions both utilize a similar enveloping technique (i.e., allowing adversarially controlled-bits and fixing $n$ bits of the input). However, the setting is different, since the former uses the dedicated key-input model. This leads to a difference in the analyses. Also the goal of [15] is only MAC preservation while ours is MPP.

Explicitly keyed compression and hash functions have both advantages and disadvantages compared to the current, unkeyed setting. The disadvantage is that none currently exist. Since we foresee new compression functions being built anyway, perhaps this is not critical; one could recommend that new compression and hash functions be keyed. However, we would expect this to cost in efficiency, since each application of the compression function must now process an additional $k$-bit input. The advantage of explicitly keyed compression and hash functions is that preservation of key-involving properties like PRF and MAC becomes easier. Indeed, MAC preservation is a goal that seems very difficult to achieve in the current no explicit key setting, at least with transforms of reasonable efficiency, and is not one we have targeted. (The reason is that if you key a compression function via the chaining variable or data input, the property of its being a MAC does not guarantee that the output has the randomness properties sufficient to be used again as a key.) One should note that the lack of MAC preservation in transforms does not mean we don't get hash functions that can be used as MACs, because we do have PRF preservation, so, assuming the compression function is a PRF, the hash function is a PRF and hence a MAC. But the MAC property of the hash function relies on the assumption that the compression function is a PRF, while in the setting of [1, 15] it only relies on the assumption that the compression function is a MAC.