

Building Application-Agile Hash Functions: the MCM Construction

Abstract

Hash functions are often expected to provide security across applications, even if there is no formal backing for these expectations. For example SHA-1 is used variously as a collision-resistant hash function and as a real-world instantiation of a random oracle; recent attacks make either use less palatable. Better security would be provided by provable collision-resistance (resting on some underlying computational hardness assumption) and, simultaneously, some guarantee of random-oracle-like behavior. We call a hash function achieving these goals *application agile*. Unfortunately, known provably CR hash functions do not typically meet both goals, as the underlying structure that allows for provable collision-resistance negates any hope of behaving like a random oracle. This paper begins the investigation of application-agile hashing, and offers a generic construction for building such objects. Our MCM construction, applied to any provably CR hash function with good regularity properties, produces the *first* hash function simultaneously provably CR in the standard model and indistinguishable from a random oracle in the ideal cipher model.

Keywords: Hash functions, random oracle, collision-resistance, pseudorandom oracles, indistinguishability.

1 Introduction

BACKGROUND. Current practice sees hash functions used in myriad ways. For example, SHA-1 is simultaneously standardized for use as a collision-resistant (CR) function (FIPS 180-1 [17]), and used to instantiate a random oracle (RSA PKCS#1 v.2.1 [25]). Ideally, a hash function utilized in such disparate ways would be application agile: it would faithfully deliver the functionality assumed by the application, under reasonable application-specific assumptions. In this work we'll use the term *application-agile hash function* to mean one that is simultaneously collision resistant under one set of assumptions, and that “behaves” like a random oracle under another.¹ The recent collision-finding attacks against SHA-1 and related hash functions [27, 28] have underscored the point that most in-use functions are not application agile (or even CR).

Several results [9, 15, 18, 24, 11] address half of our problem, namely by building provably CR hash functions. As a simple example, consider the function $H(m) = x^m \pmod n$ where x is some fixed base and n is a (supposedly) hard-to-factor composite [18, 24]. The function is provably CR since the ability to find collisions against this function implies the ability to efficiently factor n . However, such a hash function is *not* application agile, particularly because it is easily distinguishable from a random oracle. One easy distinguishing feature is that $H(2m) \equiv H(m)^2 \pmod n$, which is only true with exceedingly low probability if H is replaced by a true random oracle. The very structure that gives H and other (more efficient) provably CR functions their collision-resistance renders them poor choices for application-agile hashing [9, 26].

Other recent results [10, 3, 8] address the other half of our problem by offering constructions that “behave” like random oracles. But these results only hold within idealized models of their building blocks, and specifically fail to be CR when one steps outside of the models. We'll say more in a moment.

¹Although the definition of application agility is stretchable, and can in general include other properties.

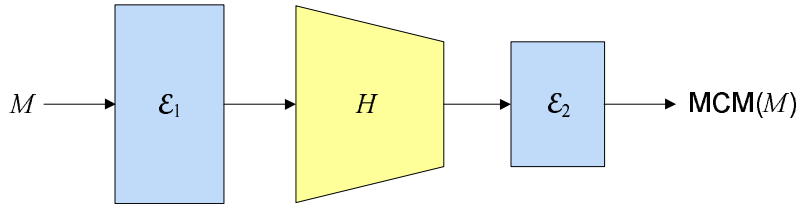


Figure 1: The MCM construction: H is a collision resistant hash function, and $\mathcal{E}_1, \mathcal{E}_2$ are mixing functions. All three components of MCM must be deterministic and publically computable.

THIS PAPER. We begin an investigation into methods for building application-agile hash functions. We first point out that none of the known hash function design approaches lead to such an object. To address this lack, we offer a generic construction that we call Mix-Compress-Mix, or MCM. See Figure 1.

The MCM construction is simple: first apply a “mixing” step \mathcal{E}_1 , then compress the result using a provably CR function H , and finally apply a second “mixing” step \mathcal{E}_2 . Here H and \mathcal{E}_1 accept variable-input-lengths. Note that \mathcal{E}_1 and \mathcal{E}_2 are necessarily deterministic, publically-computable functions. By further requiring that \mathcal{E}_1 and \mathcal{E}_2 be injective, we straightforwardly have that collisions against MCM imply collisions against H . We stress that *no* cryptographic assumptions about the mixing steps are needed for collision-resistance; simple injectivity is enough.

At the same time, MCM behaves like a random oracle when $\mathcal{E}_1, \mathcal{E}_2$ are *pseudorandom injective oracles* (PRIO), and the CR hash function is close to regular (i.e., the preimage set of any particular output isn’t too large). We’ll define formally what is a PRIO in a bit, but loosely we mean that \mathcal{E}_1 and \mathcal{E}_2 ought to behave like a random oracle that observes injectivity. To make precise the word “behaves”, we utilize the indistinguishability framework of Maurer et al. [16]. We will show that MCM is indistinguishable from a random oracle; thus, it is a pseudorandom oracle (PRO). We go on to specify the TE construction as an instantiation of the mixing steps, and show that TE achieves our PRIO security goal in the ideal cipher model.

The intuition for why MCM works to achieve application agility is relatively clear: the mixing steps obfuscate any input-output relationships of the underlying compressing step. Recall our provably CR example $H(m) = x^m \bmod n$ and the associated distinguishing attack. Adapting that attack for use against $\mathcal{H}(M) = \mathcal{E}_2(H(\mathcal{E}_1(M)))$ requires that the adversary determine non-trivial input-output relationships across both \mathcal{E}_1 and \mathcal{E}_2 .

ALTERNATIVES THAT FALL SHORT. We discuss why other seemingly natural approaches for building application-agile hash functions do not work. First we make concrete our discussion above. We say that a hash function \mathcal{H} is application-agile if it is simultaneously (1) provably collision-resistant in the standard model and (2) provably a pseudorandom oracle in the ideal cipher model. Note that (2) is only possible in an idealized setting (although not necessarily the ideal cipher model), due to well-known impossibility results [1, 7]. Relatedly (and perhaps subtly), being a PRO in an ideal setting is by itself insufficient for arguing collision-resistance in the standard model. (See [3] for more discussion.)

One might be tempted to utilize the following approach to build an application-agile hash function \mathcal{H} . The recent results from Coron et al. [10], Chang et al. [8], and Bellare and Ristenpart [3] provide domain extension transforms for building PROs from suitable compression functions. The first

two works, in particular, turn blockcipher-based compression functions from [19, 5] into hash functions that meet requirement (2). Additionally, the results of Black et al. [5] show that these blockcipher-based compression functions are CR, and so one might be tempted to conclude that requirement (1) is also met. However, the Black et al. [5] results hold only in the ideal cipher model. No standard model proofs of collision resistance for these schemes are known.

One might think that there are immediate ways of building an application-agile hash function from a provably CR function H and a blockcipher. For example, first build a PRO \mathcal{F} from the blockcipher, and then hash by computing $\mathcal{H}(M) = \mathcal{F}(H(M))$. But Coron et al. [10] point out that an adversary can easily differentiate this construction from a true monolithic random oracle. Moreover, in the standard model where \mathcal{F} ‘loses’ its idealness and is just a complexity-theoretic object, the construction is not even guaranteed to be CR. The features of the MCM construction are all apparently necessary: dropping the injectivity requirement on \mathcal{E}_1 or \mathcal{E}_2 means a loss of the collision-resistance guarantee in the standard model; omitting \mathcal{E}_1 results in an object easily differentiable from a random oracle. See Section 4 for a detailed discussion.

A NEW DESIGN PARADIGM. We point out that one could build an application-agile hash function from a multi-property-preserving transform such as EMD [3] applied to an application-agile compression function. However this leaves open the question of how to build such a compression function, and in fact no such compression functions have appeared. The MCM approach goes a different way, following the example of generic composition results from the authenticated-encryption literature [2]. Specifically, MCM allows the following clean separation of design tasks. First, design a function with strong guarantees of collision-resistance, inducing whatever structure is necessary. Second, design an injective function that destroys any structure present in its input. The MCM construction and corresponding composition theorem then gives a way to securely combine these primitives into an application-agile hash function.

THE TE CONSTRUCTION. We now turn to how to build \mathcal{E}_1 and \mathcal{E}_2 . The first step is to define the ideal behavior of these mixing steps. To this end, we formalize the notion of an *ideal injection*, which is much like a random oracle except that it is injective. Note that injectivity does not necessarily imply easy-to-invert. An ideal injection is different from an ideal cipher. The latter object is intended to capture the blackbox behavior of a blockcipher, which is a bijective mapping for each key. On the other hand, an ideal injection is not keyed, potentially handles messages of varying lengths, and can have (many) range points to which no domain points are mapped. We say a construction is a pseudorandom injective oracle (PRIO) if it is indistinguishable from an ideal injection. Note that our PRIO security definition does not give an inverse oracle for the injection. (One could do so if they wanted to model some setting in which it should be easy to invert the injection. This won’t be our case.)

We present the Tag-and-Encipher (TE) construction and show that it instantiates a PRIO. Given a blockcipher and a trapdoor one-way permutation, the TE construction specifies how to build an injective function that is indistinguishable from an ideal injection when the blockcipher is modeled as ideal. As the first construction to achieve the new PRIO goal, we view the TE construction as a proof-of-concept. It is length-increasing (outputs are larger than the inputs by the number of key bits of the underlying blockcipher), meaning that output hash values will be slightly larger compared to the outputs of H . Further, we make no claims of efficiency, as it requires two passes over the data and the application of a trapdoor permutation. In settings where speed is not essential (e.g., contract signing), the extra expense of using TE over that already incurred by hashing with a standard-model collision-resistant function H might not be prohibitive. We also offer a few suggestions toward potential speed improvements, in Section 5. Downsides having been said, the TE construction *does* show that application-agile hashing, and in particular the MCM approach, is feasible. Hopefully it also leads to

future improvements.

APPLICATION AGILE VS. APPLICATION SPECIFIC. There is an alternative approach to achieving this same goal of multi-functionality, namely to provide a toolbox of application-specific hash functions: one for collision resistance, one for instantiating a random oracle, etc. This approach has the benefit that it allows one to tailor individual functions for CR, PRO, target CR, etc. This might result in more elegant constructions and, perhaps, better property-by-property security guarantees when compared with constructions attempting to “do it all”. On the other hand, the toolbox approach requires correct and efficient implementations of several functions. It also demands that protocol designers and security practitioners be both aware of the hash properties required by their application, and able to correctly match these with the the right hash function from the toolbox. This lends itself to misuse, as the difference among security notions for hashing can be quite subtle (eg., TCR vs. second-preimage resistance). Moreover, as mentioned already, the toolbox approach does not match common practice, in which MD5 or SHA-1 (or some variant) gets used regardless of the required properties. Indeed, the component-orientated approach of MCM allows us to partially reclaim one of the benefits of the “toolbox” approach (i.e., tailoring functions to specific tasks) while still enabling application-agility.

2 Preliminaries

BASICS. Let $X, Y \in \{0, 1\}^*$. We denote the concatenation of X and Y by $X || Y$ or simply XY . The i^{th} bit of X is $X[i]$ and so $X = X[1]X[2] \cdots X[|X|]$. We write $X|_n$ (resp. $X^{[n]}$) to represent the substring consisting of the last (resp. first) n bits of X for any $n \leq |X|$. For a set S we often write $S \stackrel{\leftarrow}{\leftarrow} x$, which means $S \leftarrow S \cup \{x\}$.

Following [6, 10] we utilize interactive Turing machines for our computational model. Cryptographic primitives, schemes, and adversaries are all interactive Turing machines.

RANDOM FUNCTIONS AND INJECTIONS. Let Dom and Rng be sets. Recall that a function $f: Dom \rightarrow Rng$ is injective if $f(X) = f(X')$ implies that $X = X'$. (Necessarily for an injection $|Dom| \leq |Rng|$.) For simplicity, we only consider injections with constant stretch τ . Particularly if $X \in Dom$ then $|f(X)| = |X| + \tau$. The following algorithms implement a *random function* and a *random injection*.

<p>Algorithm $\mathbf{RF}_{Dom, Rng}(X)$: If $\mathbf{R}[X] \neq \perp$ then Ret $\mathbf{R}[X]$ Ret $\mathbf{R}[X] \stackrel{\\$}{\leftarrow} Rng$</p>	<p>Algorithm $\mathbf{RI}_{Dom, Rng}(X)$: $\ell \leftarrow X + \tau$ If $\mathbf{I}[X] \neq \perp$ then Ret $\mathbf{I}[X]$ $\mathbf{I}[X] \stackrel{\\$}{\leftarrow} \{0, 1\}^\ell \setminus \mathcal{R}_\ell$ $\mathcal{R}_\ell \stackrel{\leftarrow}{\leftarrow} \mathbf{I}[X]$ Ret $\mathbf{I}[X]$</p>
---	--

The tables \mathbf{R} and \mathbf{I} are initially everywhere set to \perp and the set \mathcal{R}_ℓ is initially empty for every ℓ . We write $f = \mathbf{RF}_{Dom, Rng}$ to signify that f is an ITM mapping points from Dom to Rng according to the algorithm specified above. We write $\mathbf{RF}_{d,r}$ if $Dom = \{0, 1\}^d$ and $Rng = \{0, 1\}^r$ for some numbers d, r . We write $\mathcal{I} = \mathbf{RI}_{Dom, Rng}$ for an ITM mapping points from Dom to Rng as per the algorithm specified above. (The other notational conventions lift to \mathbf{RI} in the obvious ways.) A random oracle is a random function that is publically accessible by all parties. Similarly an ideal injection is a publically-accessible random injection.

IDEAL CIPHERS. A blockcipher $E: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ is a function for which $E(K, \cdot) = E_K(\cdot)$ is a permutation for every $K \in \{0, 1\}^k$. The inverse of E is D and is defined such that $D(K, Y) = M$ iff $E(K, M) = Y$. An ideal cipher is a blockcipher uniformly selected from $\mathbf{BC}(k, n)$, the space of all blockciphers with k -bit keys and n -bit blocksize. In the ideal cipher model, both an ideal cipher E and its inverse are given to all parties as oracles.

SECURITY NOTIONS. Let $f: \mathcal{K} \times \text{Dom} \rightarrow \text{Rng}$ be a function family indexed by a non-empty key space \mathcal{K} . Then we define the collision-finding advantage of an adversary \mathcal{A} against f as

$$\text{Adv}_f^{\text{cr}}(\mathcal{A}) = \Pr \left[K \xleftarrow{\$} \mathcal{K}; (X, X') \xleftarrow{\$} \mathcal{A}(K) : f_K(X) = f_K(X') \right]$$

where the probability is over the random choice of K and the random coins utilized by \mathcal{A} .

Recall that a function $f: \text{Dom} \rightarrow \{0, 1\}^\eta$ is *regular* if each image has an equal number of preimages. A function family $f: \mathcal{K} \times \text{Dom} \rightarrow \{0, 1\}^\eta$ is regular if f_K is regular for each $K \in \mathcal{K}$. Define $\text{Prelm}(K, \ell, Y)$ for any $K \in \mathcal{K}$, ℓ such that $\{0, 1\}^\ell \subseteq \text{Dom}$, and $Y \in \{0, 1\}^\eta$ to be the the set of preimages (under K) of Y that are of length ℓ : $\text{Prelm}(K, \ell, Y) = \{X : X \in \text{Dom} \wedge |X| = \ell \wedge f_K(X) = Y\}$. Further we define the following function related to f

$$\delta(K, \ell, Y) = \frac{|\text{Prelm}(K, \ell, Y)| - 2^{\ell-\eta}}{2^\ell}.$$

The δ function is a simple measure of, proportionally, how many more (or less) preimages are claimed by an image point than would be if f_K were regular. Let $\Delta_K = \max\{\delta(K, \ell, Y)\}$ over all choices of ℓ and Y . Then we say a function family f is Δ -regular if

$$\frac{\sum_{K \in \mathcal{K}} \Delta_K}{|\mathcal{K}|} \leq \Delta.$$

Intuitively Δ is a simple measure of how far, on average, a random instance of f is from being regular.

We follow the formalization of indifferentiability from [10, 3]. A *simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_l)$ is an interactive Turing machine with l interfaces $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_l$. Let C be some cryptographic scheme utilizing primitives f_1, \dots, f_l and let Dom and Rng be sets. We define the pro and prio advantages of an adversary \mathcal{A} against C as

$$\begin{aligned} \text{Adv}_{C, \mathcal{S}}^{\text{pro}}(\mathcal{A}) &= \Pr \left[\mathcal{A}^{C^{f_1, \dots, f_l}, f_1, \dots, f_l} \Rightarrow 1 \right] - \Pr \left[\mathcal{A}^{\mathcal{F}, \mathcal{S}^{\mathcal{F}}} \Rightarrow 1 \right] \\ \text{Adv}_{C, \mathcal{S}}^{\text{prio}}(\mathcal{A}) &= \Pr \left[\mathcal{A}^{C^{f_1, \dots, f_l}, f_1, \dots, f_l} \Rightarrow 1 \right] - \Pr \left[\mathcal{A}^{\mathcal{I}, \mathcal{S}^{\mathcal{I}}} \Rightarrow 1 \right] \end{aligned}$$

where $\mathcal{F} = \text{RF}_{\text{Dom}, \text{Rng}}$ and $\mathcal{I} = \text{RI}_{\text{Dom}, \text{Rng}}$ and the probabilities are over the random coins used by the appropriate objects. We disallow \mathcal{A} from making *pointless* queries, which in this setting means querying an oracle twice.

Informally we call a cryptographic scheme C a pseudorandom oracle (PRO) if there exists an “efficient” simulator against which all adversaries have “small” pro advantage. Likewise we call a cryptographic scheme C a pseudorandom injective oracle (PRIO) if there exists an “efficient” simulator against which all adversaries have “small” prio advantage. We do not formalize “efficient” or “small”, giving concrete running times and bounds, instead. We define $\text{Time}_f(\mu)$ as the worst-case time to compute f on a message of length at most μ .

3 The MCM Construction

Fix numbers η and τ . Let $H: \mathcal{K} \times \mathcal{M}_H \rightarrow \{0, 1\}^\eta$ be a function family with key space \mathcal{K} and domain $\mathcal{M}_H = \{0, 1\}^{\leq L}$ for some large number L (e.g., 2^{64}). Let $\mathcal{E}_1: \mathcal{M} \rightarrow \mathcal{M}_H$ be an injective function where $\mathcal{M} = \{0, 1\}^{\leq L'}$ for $L' = L - \tau$. For any $X \in \mathcal{M}$ we have that $|\mathcal{E}_1(X)| = |X| + \tau$, hence τ is the *stretch* of \mathcal{E}_1 . Finally let $\mathcal{E}_2: \{0, 1\}^\eta \rightarrow \{0, 1\}^{\eta+\tau}$ be an injective function. Then we define the hash function $\mathcal{H} = \text{MCM}[\mathcal{E}_1, H, \mathcal{E}_2]$ with key space \mathcal{K} , domain \mathcal{M} , and range $\{0, 1\}^{\eta+\tau}$ by $\mathcal{H}_K(M) = \mathcal{H}(K, M) = \mathcal{E}_2(H_K(\mathcal{E}_1(M)))$. Overloading our notation, if $\mathcal{I}_1 = \text{RI}_{\mathcal{M}, \mathcal{M}_H}$ and $\mathcal{I}_2 = \text{RI}_{\eta, \eta+\tau}$

then we write $\mathcal{H} = \text{MCM}[\mathcal{I}_1, H, \mathcal{I}_2]$ where now \mathcal{H} is itself an ITM using oracle access to \mathcal{I}_1 and \mathcal{I}_2 to calculate $\mathcal{H}_K(M) = \mathcal{I}_2(H_K(\mathcal{I}_1(M)))$.

Here τ is also the stretch of \mathcal{H} — it's the number of bits beyond η needed to hold a hash value. Ideally $\tau = 0$, in which case \mathcal{E}_1 and \mathcal{E}_2 would be a permutations. We have the following theorem, which states that \mathcal{H} inherits the collision-resistance of H .

Theorem 3.1 *Let $H: \mathcal{K} \times \mathcal{M}_H \rightarrow \{0, 1\}^\eta$ be a function and $\mathcal{E}_1: \mathcal{M} \rightarrow \mathcal{M}_H$ and $\mathcal{E}_2: \{0, 1\}^\eta \rightarrow \{0, 1\}^{\eta+\tau}$ be injections. Let $\mathcal{H} = \text{MCM}[\mathcal{E}_1, H, \mathcal{E}_2]$. Let \mathcal{A} be an adversary that runs in time t and outputs messages each of length at most μ . Then there exists an adversary \mathcal{B} such that*

$$\text{Adv}_{\mathcal{H}}^{\text{cr}}(\mathcal{A}) = \text{Adv}_H^{\text{cr}}(\mathcal{B})$$

where \mathcal{B} runs in time $t' \leq t + 2(c\mu + \text{Time}_{\mathcal{E}_1}(\mu))$ for some absolute constant c . \square

Proof: Let \mathcal{B} be the adversary that behaves as follows. It runs \mathcal{A} , which eventually outputs (X, X') . Then \mathcal{B} outputs $(\mathcal{E}_1(X), \mathcal{E}_1(X'))$. We have that if $\mathcal{H}(X) = \mathcal{H}(X')$ then because \mathcal{E}_1 and \mathcal{E}_2 are injections, necessarily $H(\mathcal{E}_1(X)) = H(\mathcal{E}_1(X'))$. Adversary \mathcal{B} runs in time $t' \leq t + 2(c\mu + \text{Time}_{\mathcal{E}_1}(\mu))$ where c is an absolute constant. \blacksquare

We point out that similar theorems could be made for related collision-resistance properties, such as target collision-resistance, second preimage resistance, and other such variants. The next theorem captures that MCM is a PRO if both \mathcal{E}_1 and \mathcal{E}_2 are PRIOs.

Theorem 3.2 *Let $H: \mathcal{K} \times \mathcal{M}_H \rightarrow \{0, 1\}^\eta$ be a Δ -regular function, $\mathcal{I}_1 = \text{RI}_{\mathcal{M}, \mathcal{M}_H}$, and $\mathcal{I}_2 = \text{RI}_{\eta, \eta+\tau}$. Let $\mathcal{H} = \text{MCM}[\mathcal{I}_1, H, \mathcal{I}_2]$. Let ν be the minimal message length of \mathcal{H} . Let \mathcal{A} be an adversary that runs in time t and making at most (q_1, q_2, q_3) queries with the combined length of all queries being at most μ . Then there exists an adversary \mathcal{B} such that*

$$\text{Adv}_{\mathcal{H}, \mathcal{S}}^{\text{pro}}(\mathcal{A}) \leq \text{Adv}_H^{\text{cr}}(\mathcal{B}) + \frac{(q_1 + q_3)^2}{2^{\eta+\tau}} + \frac{(q_1 + q_2)^2}{2^{\nu+\tau}} + \frac{q_1 q_3}{2^\eta} + q_1 q_3 \Delta$$

where the simulator \mathcal{S} , specified below in the proof, runs in time $t_{\mathcal{S}} \leq c\mu(q_1 + q_1 q_3)$ for some absolute constant c and makes at most $\min\{q_2, q_3\}$ oracle queries. Adversary \mathcal{B} runs in time at most $t_{\mathcal{B}} \leq t + t_{\mathcal{S}} + c'\mu$ for some absolute constant c' . \square

Before the proof, we discuss the theorem statement. As long as \mathcal{E}_1 and \mathcal{E}_2 are PRIOs we can securely replace them by actual ideal injections (as per the composition theorem of [16]). Then, Theorem 3.2 states that no adversary can differentiate between a real random oracle and the construction unless it is given sufficient time to break the collision-resistance of H or allowed to make approximately $2^{(\tau + \min\{\eta, \nu\})/2}$ queries. Here ν could in fact be small, since this is the minimal message length in the domain of our hash function (and we'd certainly want to include short messages). However, in practice, H will have some minimal message length ν_H (e.g., the blocksize of an underlying compression function) to which short messages would necessarily be padded anyway. Thus, \mathcal{H} can ‘aggressively’ pad short strings to a minimal length $\nu = \nu_H - \tau$, recovering our security guarantee.

Proof: First we fix a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$. The first interface simply implements a random injection $\mathcal{I}_1 = \text{RI}_{\mathcal{M}, \mathcal{M}_H}$ and the second interface behaves as follows (note that since state is shared, this interface sees the internal state of \mathcal{I}_1).

procedure $\mathcal{S}_2(Y)$

If $\exists M$ s.t. $Y = H_K(\mathcal{I}_1(M))$ then Ret $\mathbf{F}[Y] \leftarrow \mathcal{R}(M)$

Ret $\mathbf{F}[Y] \xleftarrow{\$} \{0, 1\}^\eta$

<p>procedure $\mathcal{O}_1(M)$ $j \leftarrow j + 1$; Ret $MCMSub(M, j)$</p> <p>subroutine $MCMSub(M, j)$ If $\exists i < j$ s.t. $M^i = M^j$ then Ret C^i $X^j \leftarrow \text{RI}_{\mathcal{M}, \mathcal{M}_H}(M^j)$ $Y^j \leftarrow H(X^j)$; $C^j \stackrel{\\$}{\leftarrow} \{0, 1\}^\eta$ If $F[Y^j] \neq \perp$ then bad \leftarrow true; $C^j \leftarrow F[Y^j]$ Ret $F[Y^j] \leftarrow C^j$</p>	<p>procedure $\mathcal{O}_2(M_s)$ G0 G1 $j \leftarrow j + 1$; $M_s^j \leftarrow M_s$; Ret $X_s^j \leftarrow \text{RI}_{\mathcal{M}, \mathcal{M}_H}(M_s^j)$</p> <p>procedure $\mathcal{O}_3(Y_s)$ $j \leftarrow j + 1$ If $\exists i < j$ s.t. $Y_s = H_K(X_s^i)$ then Ret $MCMSub(M_s^i, j)$ $C_s^i \stackrel{\\$}{\leftarrow} \{0, 1\}^{\eta+\tau}$ If $F[Y_s^i] \neq \perp$ then bad \leftarrow true; $C_s^j \leftarrow F[Y_s^j]$ Ret $F[Y_s^i] \leftarrow C_s^j$</p>
<p>procedure $\mathcal{O}_1(M)$ $j \leftarrow j + 1$; Ret $MCMSub(M, j)$</p> <p>subroutine $MCMSub(M, j)$ If $\exists i < j$ s.t. $M^i = M^j$ then Ret C^i $X^j \leftarrow \text{RI}_{\mathcal{M}, \mathcal{M}_H}(M^j)$ $Y^j \leftarrow H(X^j)$ If $\exists i$ s.t. $Y^i = Y^j$ then bad1 \leftarrow true If $\exists i$ s.t. $Y_s^i = Y^j$ then bad2 \leftarrow true Ret $C^j \stackrel{\\$}{\leftarrow} \{0, 1\}^{\eta+\tau}$</p>	<p>procedure $\mathcal{O}_2(M_s)$ G2 $j \leftarrow j + 1$; $M_s^j \leftarrow M_s$; Ret $X_s^j \leftarrow \text{RI}_{\mathcal{M}, \mathcal{M}_H}(M_s^j)$</p> <p>procedure $\mathcal{O}_3(Y_s)$ $j \leftarrow j + 1$ If $\exists i < j$ s.t. $Y_s^j = H_K(X_s^i)$ then Ret $MCMSub(M_s^i, j)$ $Y_s^j \stackrel{\\$}{\leftarrow} Y_s$ If $\exists i$ s.t. $Y^i = Y_s^j$ then bad2 \leftarrow true Ret $C_s^j \stackrel{\\$}{\leftarrow} \{0, 1\}^{\eta+\tau}$</p>

Figure 2: Games used in the proof of Theorem 3.2. Initially $j = 0$, the table F is everywhere \perp , and the multiset \mathcal{DF} is empty.

This interface simply checks if \mathcal{I}_1 already maps a string M to a preimage under H_K of the queried value Y . If multiple such M exist, then one is used arbitrarily. We must now bound the probability that an adversary \mathcal{A} can distinguish between the sets of oracles $(\mathcal{H}, \mathcal{I}_1, \mathcal{I}_2)$ and $(\mathcal{R}, \mathcal{S}_1, \mathcal{S}_2)$ where $\mathcal{I}_1, \mathcal{I}_2$ are as defined in the theorem statement and $\mathcal{R} = \text{RF}_{\mathcal{M}, \eta+\tau}$.

We utilize a hybrid argument using the following sets of oracles.

- (1) $(\mathcal{H}, \mathcal{I}_1, \mathcal{I}_2)$
- (2) $(\mathcal{H}, \mathcal{I}_1, \mathcal{F})$
- (3) $(\mathcal{R}, \mathcal{S}_1, \mathcal{S}_{\mathcal{E}_2})$

where \mathcal{H} always utilizes the second and third oracles to compute its response, $\mathcal{I}_1 = \text{RI}_{\mathcal{M}, \mathcal{M}_H}$, $\mathcal{I}_2 = \text{RI}_{\eta, \eta+\tau}$, $\mathcal{R} = \text{RF}_{\mathcal{M}, \eta+\tau}$, and $\mathcal{F} = \text{RF}_{\eta, \eta+\tau}$. Letting $p_{ij} = \Pr[\mathcal{A}^{(\cdot)_i} \Rightarrow 1] - \Pr[\mathcal{A}^{(\cdot)_j} \Rightarrow 1]$ where $(\cdot)_i$ is replaced by the oracles marked by i in the list above. Then $\text{Adv}_{\mathcal{H}, \mathcal{S}}^{\text{pro}}(\mathcal{A}) \leq p_{12} + p_{23}$. We have that $p_{12} \leq (q_1 + q_3)^2 / 2^{\eta+\tau}$ by a straightforward birthday bound. To bound p_{23} we utilize games [4].

Game G0 is shown in Figure 2 (boxed statements included). G0 implements the oracles $(\mathcal{H}, \mathcal{I}_1, \mathcal{F})$. Game G1 is exactly like G0 but with the boxed statements removed. It implements the oracles $(\mathcal{H}, \mathcal{I}_1, \mathcal{S}_{\mathcal{E}_2})$. The games are identical-until-bad and so by the fundamental lemma of game-playing [4]

$$p_{23} \leq \Pr[\mathcal{A}^{\text{G1}} \text{ sets bad}].$$

Game G2, shown in Figure 2, is a conservative modification of game G1. Instead of setting **bad** we set either **bad1** or **bad2**. The former is set when two values Y^i, Y^j chosen within $MCMSub$ collide. The latter is set when a Y^i value chosen within $MCMSub$ collides with a Y_s^j value queried to \mathcal{O}_3 (but for which the conditional in \mathcal{O}_3 did not evaluate to true). We have that

$$\begin{aligned} \Pr[\mathcal{A}^{\text{G1}} \text{ sets bad}] &= \Pr[\mathcal{A}^{\text{G2}} \text{ sets bad1} \vee \mathcal{A}^{\text{G2}} \text{ sets bad2}] \\ &\leq \Pr[\mathcal{A}^{\text{G2}} \text{ sets bad1}] + \Pr[\mathcal{A}^{\text{G2}} \text{ sets bad2}]. \end{aligned}$$

We can bound the probability of `bad1` being set as follows. We build an adversary \mathcal{B} which breaks the collision-resistance of H whenever `bad1` is set in $G2$. Formally, let \mathcal{B} be the adversary that, on input K , runs $G2^{\mathcal{A}}$ using K where appropriate in the course of the execution. If ever two values $X^i \neq X^j$ are computed such that $H_K(X^i) = H_K(X^j)$ then \mathcal{B} returns (X^i, X^j) . Now to show that $\Pr[\mathcal{A}^{G2} \text{ sets bad1}] = \mathbf{Adv}_H^{\text{cr}}(\mathcal{B})$. First note that the distributions of random variables in \mathcal{B} and in $G2^{\mathcal{A}}$ are the same by construction. If \mathcal{A} ever forces two values Y^i and Y^j to be equal we see that by the structure of $G2$ necessarily $M^i \neq M^j$ because otherwise the first if statement in $MCMSub$ would have evaluated to true. Since RI is injective, $M^i \neq M^j$ implies $X^i \neq X^j$ and so anytime `bad1` is set \mathcal{B} succeeds.

To bound the probability of `bad2` being set, we first perform one more game transition. Game $G3$ is exactly like $G2$ except we replace both uses of $\text{RI}_{\mathcal{M}, \mathcal{M}_H}$ with $\text{RF}_{\mathcal{M}, \mathcal{M}_H}$. We have that

$$\Pr[G2^{\mathcal{A}} \text{ sets bad2}] \leq \Pr[G3^{\mathcal{A}} \text{ sets bad2}] + \frac{(q_1 + q_2)^2}{2^{\nu+\tau}}$$

(which follows via a straightforward application of the fundamental lemma of game playing — we omit the details). Now we bound the probability that `bad2` is set in $G3$. If $Y^i = Y_s^j$ for some i, j , then this implies that $H_K(X^i) = Y_s^j$ and necessarily X^i and Y^i were set due to an \mathcal{O}_1 query. At query j the adversary must not have previously queried \mathcal{O}_1 on the string M^i or queried \mathcal{O}_2 on the string X^i , since otherwise the value Y_s^j would never have been defined (only Y_s , and *not* Y_s^j would be defined). Thus the adversary has no information about X^i when it chooses Y_s^j . We have that $\Pr[G3^{\mathcal{A}} \text{ sets bad2}] = \Pr[H_K(X^i) = Y_s^j \text{ for some } i, j]$ where the latter probability is over the choice of K and choices of X^i . Assume that \mathcal{A} can always pick q_1 message lengths $|X^i|$ and q_3 values Y_s^j that maximize the above probability for a key K . Notate these maximizing lengths by $1_1^K, \dots, 1_{q_1}^K$ and values by $Y_1^K, \dots, Y_{q_3}^K$. Then we have that

$$\begin{aligned} \Pr[H_K(X^i) = Y_s^j \text{ for some } i, j] &\leq \sum_{i=1}^{q_1} \sum_{K \in \mathcal{K}} \Pr[H_K(X_i) \in \{Y_1, \dots, Y_{q_3}\} | K = K] \cdot \Pr[K = K] \\ &= \frac{1}{|\mathcal{K}|} \sum_{i=1}^{q_1} \sum_{K \in \mathcal{K}} \Pr \left[\bigvee_{i=1}^{q_3} X_i \in \text{Prelm}(K, 1_i, Y_1) \right] \\ &\leq \frac{1}{|\mathcal{K}|} \sum_{i=1}^{q_1} \sum_{K \in \mathcal{K}} \sum_{j=1}^{q_3} \Pr[X_i \in \text{Prelm}(K, 1_i, Y_j)] \\ &= \frac{1}{|\mathcal{K}|} \sum_{i=1}^{q_1} \sum_{j=1}^{q_3} \sum_{K \in \mathcal{K}} \frac{|\text{Prelm}(K, 1_i, Y_j)|}{2^{1_i}} \\ &= \frac{1}{|\mathcal{K}|} \sum_{i=1}^{q_1} \sum_{j=1}^{q_3} \sum_{K \in \mathcal{K}} \frac{2^{1_i - \eta}}{2^{1_i}} + \delta(K, 1_i, Y_j) \\ &= \sum_{i=1}^{q_1} \sum_{j=1}^{q_3} \left[\frac{2^{1_i - \eta}}{2^{1_i}} + \sum_{K \in \mathcal{K}} \frac{\delta(K, 1_i, Y_j)}{|\mathcal{K}|} \right] \\ &\leq \frac{q_1 q_3}{2^\eta} + \sum_{i=1}^{q_1} \sum_{j=1}^{q_3} \sum_{K \in \mathcal{K}} \frac{\Delta_K}{|\mathcal{K}|} = \frac{q_1 q_3}{2^\eta} + q_1 q_3 \Delta \end{aligned}$$

where X_i is always a random selection of 1_i bits. The theorem statement follows. \blacksquare

4 Insecurity of Other Approaches

Here we give just a brief investigation of several alternative approaches to application-agile hashing, including potential simplifications of MCM. In all cases, either the resulting object is not provably collision-resistant in the standard model or not provably a PRO in an ideal model.

USING EXISTING BLOCKCIPHER-BASED HASH FUNCTIONS. Let $E: \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a blockcipher, modeled as ideal. Let f be a $2n$ -bit to n -bit compression function. Fix some suitable domain extension transform, for example Merkle-Damgård with a prefix-free encoding. That is $\mathcal{H}(M) = f^+(g(M))$, where $f^+(M_1 \cdots M_m)$ is equal to Y_m defined recursively by $Y_0 = IV$ (some constant) and $Y_i = f(Y_{i-1}, M_i)$, and $g: \{0, 1\}^* \rightarrow (\{0, 1\}^n)^+$ is a prefix-free padding function. For simplicity let $g(M)$ simply split M into blocks of $n - 1$ bits (M having been appropriately padded), and then appending a zero to each block except the last and appending a one to the last block. If f is one of the twenty group-1/2 schemes from [5], then \mathcal{H} is collision-resistant in the ideal cipher model. Moreover, a recent paper by Chang et al. [8] shows that sixteen of these twenty yield a PRO \mathcal{H} .

However as soon as one leaves the ideal cipher model, \mathcal{H} is *not* provably CR. For example let E' be the blockcipher defined as follows:

$$E'(K, M) = \begin{cases} M & \text{if } K = 0^k \\ E(K, M) & \text{otherwise} \end{cases}$$

where, now, E is no longer ideal. Let $f(Y_{i-1}, M_i) = E'(M_i, Y_{i-1}) \oplus Y_{i-1}$. We can see that an adversary can trivially find collisions against \mathcal{H} built using E' . This is true even though E' is a good pseudorandom permutation (the usual standard model security property of blockciphers) whenever E is also.²

REMOVING INJECTIVITY REQUIREMENTS. If either \mathcal{E}_1 or \mathcal{E}_2 are not injective, then the MCM construction loses its provable collision-resistance. Assuming they are built from using blockciphers (as we suggest), then one can, in spirit similar to the counter-example above, construct a collision resistant function H' and a good PRP E' that, when utilized in MCM, would lead to a trivial collisions.

Note that one might imagine replacing \mathcal{E}_1 and \mathcal{E}_2 with objects that are not injective, yet have some other standard model guarantees to ensure provable collision-resistance in MCM. Short of establishing their collision-resistance, its not clear what properties could achieve this goal. Additionally, this approach would seem to violate the separation of design tasks intrinsic to the MCM approach.

OMITTING \mathcal{E}_1 FROM MCM. If one omits the first “mixing” step \mathcal{E}_1 of MCM, then the construction no longer results in a PRO. This result is essentially equivalent to the Coron et al. insecurity result regarding the composition of a CR and one-way function H with a random oracle [10], but we state a version of it here for completeness. Let $\mathcal{H} = \text{CM}[H, \mathcal{I}_2]$ be this modified construction for $\mathcal{I}_2 = \text{RI}_{\eta, \eta + \tau}$, i.e. $\mathcal{H}(M) = \mathcal{I}_2(H(M))$. Now we show that \mathcal{H} is easily differentiable from a true random oracle $\mathcal{R} = \text{RF}_{\mathcal{M}_H, \eta + \tau}$. Let \mathcal{A} be an adversary that queries it’s first oracle on a uniformly selected message of length $M \in \mathcal{M}_H$ of some length ℓ . Let the returned value be C . Now the adversary queries its second oracle (representing either \mathcal{I}_2 or a simulator) on $H_K(C)$. Let the returned value be C' . If $C = C'$ then \mathcal{A} returns one, guessing that it’s interacting with the construction. Otherwise it returns zero, guessing that it’s interacting with the true random oracle. We have that $\Pr[\mathcal{A}^{\mathcal{H}, \mathcal{I}_2} \Rightarrow 1] = 1$. On the other hand, $\Pr[\mathcal{A}^{\mathcal{R}, \mathcal{S}} \Rightarrow 1]$ is bounded by the advantage of a related adversary in breaking the one-wayness of H .

²Hopwood and Wagner were the first to note (in postings on sci.crypt) that one could exhibit good PRPs that would make finding collisions in the twenty [5] functions trivial.

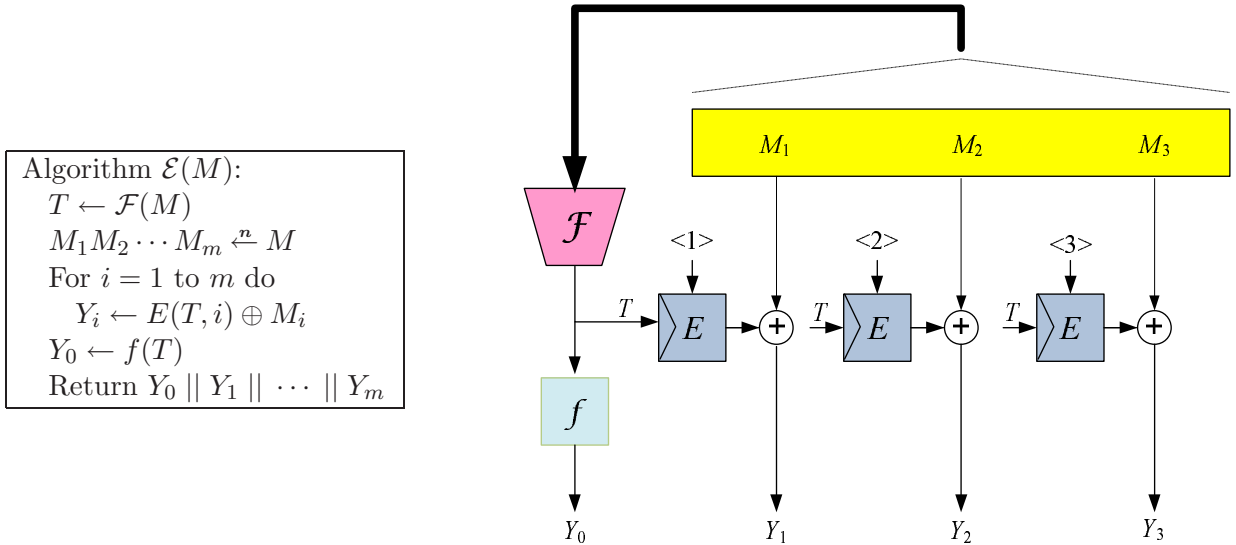


Figure 3: (Left) Algorithm $\mathcal{E} = \text{TE}[E, \mathcal{F}, f]$. (Right) A diagram of \mathcal{E} applied to a message M for which $|M| = 3n$.

5 The TE Construction

To instantiate the MCM construction, we need to implement a variable-input-length pseudorandom injection oracle (PRIO), an object never before considered. Our starting point is an ideal cipher $E: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$. Recall that both E and its inverse D is available to the adversary. While there are known secure ways of building variable-input-length deterministic ciphers from a blockcipher (for just a few examples see [14, 13, 12, 22]), those settings assume secret keys and are not publically-computable — such constructions are not applicable in our current setting.

To overcome the challenges of building such an object, our construction will employ two primitives beyond E . The first is a random oracle \mathcal{F} . By the results of [10, 8], though, we can build such an object with E .³ The second is a trapdoor one-way permutation. Recall that a trapdoor permutation is a permutation f that is one-way, but for which there exists trapdoor information that permits computing f^{-1} efficiently. The RSA and Rabin functions are conjectured to be such permutations [23, 20, 21]. Let \mathbb{F} be a trapdoor permutation generator: on input 1^k it outputs a trapdoor permutation pair (f, f^{-1}) where $f: \{0, 1\}^k \rightarrow \{0, 1\}^k$ and $f^{-1}(f(X)) = X$. We capture the one-wayness of a function generated by \mathbb{F} via the following definition

$$\text{Adv}_{\mathbb{F}}^{\text{owf}}(\mathcal{A}) = \Pr \left[(f, f^{-1}) \stackrel{\$}{\leftarrow} \mathbb{F}(1^k); X \stackrel{\$}{\leftarrow} \{0, 1\}^k; Y \leftarrow f(X); X' \stackrel{\$}{\leftarrow} \mathcal{A}(f, Y) : f(X) = f(X') \right].$$

THE TE CONSTRUCTION. Let $E: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a blockcipher, $\mathcal{F}: \mathcal{M} \rightarrow \{0, 1\}^k$ be a random oracle, and let $f: \{0, 1\}^k \rightarrow \{0, 1\}^k$ be a permutation. Then we define the injection

³To make \mathcal{F} independent of other usage of E , we can reserve a bit of the key input of E : setting it to one for use within \mathcal{F} and setting it to zero for other uses.

procedure $\mathcal{S}_E(K, C)$: If $\exists j$ s.t. $\Gamma^j = K$ and $C \leq M^j /n$ then Ret $M_C^j \oplus Y_{C+1}^j$ Ret $Y \stackrel{\$}{\leftarrow} \{0, 1\}^n$	procedure $\mathcal{S}_{\mathcal{F}}(M)$ $j \leftarrow j + 1$; $M^j \leftarrow M$ $Y_0^j \parallel \tilde{Y}^j \leftarrow \mathcal{I}(M^j)$ $\Gamma^j \leftarrow f^{-1}(Y_0^j)$ Ret Γ^j
procedure $\mathcal{S}_D(K, Y)$ Return $D \stackrel{\$}{\leftarrow} \{0, 1\}^n$	procedure $\mathcal{S}_{\text{PGen}}()$ $(f, f^{-1}) \stackrel{\$}{\leftarrow} \mathbb{F}(1^k)$ Return f

Figure 4: The simulator \mathcal{S} used in proof of Theorem 5.1. Initially $j = 0$.

$\mathcal{E} = \text{TE}[E, \mathcal{F}, f]$ by the algorithm in Figure 3. The domain of \mathcal{E} is $\{0, 1\}^{\leq L}$ where $L = n \cdot 2^{128}$. It maps a string X to a string of length $|X| + k$. Now let E be an ideal cipher, $\mathcal{F} = \text{RF}_{\mathcal{M}, k}$, and PGen be the oracle that behaves as follows. When called, it computes $(f, f^{-1}) \stackrel{\$}{\leftarrow} \mathbb{F}(1^k)$ and returns f . Subsequent calls return the same f . (This oracle idealizes a trusted setup of the public one-way permutation.) Overloading our notation, define $\mathcal{E} = \text{TE}[E, \mathcal{F}, \text{PGen}]$ to be the ITM generated by following the program of Figure 3 except it utilizes PGen to get f initially and queries E and \mathcal{F} oracles where appropriate.

The TE construction runs \mathcal{F} over the message to derive a tag, which is used as a key for a variant of counter-mode encryption [29]. The tag is then ‘hidden’ via f . Note that outputting $Y_0 = f(T)$ ensures injectivity and, in particular, omitting it in the output results in a function that is *not* injective.

Why should TE work? Generating tags via \mathcal{F} ensures that they are uniformly distributed. This means that tags are difficult to predict and rarely collide. The tag is used as a message-specific key for the CTR-mode-like encryption of the message. This means that, with high probability, the tag generates a new, uniformly random pad. The one-way permutation f is needed to hide tag values from the adversary while simultaneously ensuring the injectivity of the entire object.

Theorem 5.1 *Let $E: \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be an ideal cipher, $\mathcal{F} = \text{RF}_{\mathcal{M}, k}$, and PGen be the oracle described above. Let $\mathcal{E} = \text{TE}[E, \mathcal{F}, \text{PGen}]$. Let \mathcal{A} be an adversary that asks at most $(q_1, q_2, q_3, q_4, 1)$ oracle queries, each of length at most μ bits, and runs in time at most t . Then there exists an adversary \mathcal{C} such that*

$$\text{Adv}_{\mathcal{E}, \mathcal{S}}^{\text{prio}}(\mathcal{A}) \leq q_1 \text{Adv}_{\mathbb{F}}^{\text{owf}}(\mathcal{C}) + \frac{(q_1 \sigma + q_2 + q_3)^2}{2^n} + \frac{q_1^2 - q_1}{2^{k+1}}$$

where $\sigma = \lceil \mu/n \rceil$ and \mathcal{S} , the simulator defined in Figure 4, runs in time at most $t_{\mathcal{S}} \leq c(\mu + q_2 q_4)$ for some absolute constant c and makes $q_{\mathcal{S}} = q_4$ queries. Adversary \mathcal{C} runs in time $t' \leq t + t_{\mathcal{S}} + (q_2 + q_4) \text{Time}_f + (q_1 + q_2 + q_4) \log(q_1 + q_2 + q_4) + c'\mu$.

A full proof of the theorem is provided in Appendix A, here we just provide a brief proof sketch. An adversary is given either the oracles $(\mathcal{E}, E, D, \mathcal{F}, \text{PGen})$ or the oracles $(\mathcal{I}, \mathcal{S}_E, \mathcal{S}_D, \mathcal{S}_{\mathcal{F}}, \mathcal{S}_{\text{PGen}})$. Intuitively the structure of TE ensures that an adversary, attempting to discover information about the tag and the random pad created for some message M , must reveal to the simulator M (via the fourth oracle). Knowing M , the simulator can ‘program’ the random pad to be consistent with output of the ideal injection \mathcal{I} .

The simulator will fail if either of two events occurs. The first event corresponds to when two tags collide in the course of simulating the construction. If this happens the CTR mode must generate the same pad, and no longer hides relationships between input and output bits. Such an event will occur with low probability because \mathcal{F} is a PRO. The second kind of event is if the adversary infers a tag

value without utilizing its fourth oracle (\mathcal{F} or $\mathcal{S}_{\mathcal{F}}$). If it can do so, then it can compute the pad using the second oracle (E or $\mathcal{S}_{\mathcal{E}}$) before the simulator knows the message the tag corresponds to. This event should happen with low probability because it requires the adversary inverts f on some image returned as the first k bits of a query to the first oracle. We can bound the probability of \mathcal{A} inverting f on some point in terms of its ability to invert on a particular point (hence the $q_1 \text{Adv}_{\mathbb{F}}^{\text{owf}}(\mathcal{C})$ term).

DISCUSSION. We point out that the one-way permutation really is requisite: omitting it would result in a construction easily differentiable from a random injective oracle. An adversary could simply query its first oracle on a random message M_1 , receiving $T \parallel Y_1$. Then the adversary could query its third oracle (either D or \mathcal{S}_D) on (T, Y_1) . At this point the simulator has no knowledge about M_1 and will therefore only respond correctly with low probability.

EFFICIENCY IMPROVEMENT. When utilizing TE within MCM there exists a potential efficiency improvement. Instead of computing a new tag over the output of H in \mathcal{E}_2 , one could reuse the tag computed in \mathcal{E}_1 . This would reduce the computational work, beyond computing H , to one use of \mathcal{F} and f and the two uses of the CTR-mode-like encryption. That said, we offer no proof, and leave an analysis to future work.

6 Open Problems

This first investigation into application-agile hashing with provable collision-resistance raises numerous interesting research questions:

1. Can more efficient “mixing” steps be constructed? In particular, building a construction that is length-preserving and/or dispensing with the trapdoor one-way permutation.
2. Can general, component-orientated constructions be built that address additional applications (beyond collision resistance and “behaving” like a random oracle)?
3. Can upper bounds on Δ be proven for proposed collision resistant hash functions?

References

- [1] 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.
- [2] M. Bellare, C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. Advances in Cryptology - ASIACRYPT '00. Volume 1976 of Lecture Notes in Computer Science, Springer (2000) 531-545.
- [3] Bellare, M., Ristenpart, T.: Multi-property-preserving Hash Domain Extension and the EMD Transform In: Advances in Cryptology - ASIACRYPT '06. Vol. 4284 of Lecture Notes in Computer Science, Springer (2006) 299–314.
- [4] 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.
- [5] J. Black, P. Rogaway, T. Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV.. Advances in Cryptology - CRYPTO '02. Volume 2442 of Lecture Notes in Computer Science, Springer (2002) 320–325.
- [6] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Symposium on Foundations of Computer Science - FOCS '01. IEEE Computer Society (2001) 136–145.

- [7] Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. *J. ACM* **51**(4) (2004) 557–594.
- [8] D. Chang, S. Lee, M. Nandi, M. Yung. Indifferentiable Security Analysis of Popular Hash Functions with Prefix-Free Padding. In: *Advances in Cryptology - ASIACRYPT '06*. Volume 4284 of *Lecture Notes in Computer Science*, Springer (2006) 283–298.
- [9] S. Contini, A. Lenstra, R. Steinfeld. VSH, an Efficient and Provable Collision-Resistant Hash Function.. *Advances in Cryptology - EUROCRYPT '06*. Volume 4004 of *Lecture Notes in Computer Science*, Springer (2006) 165–182.
- [10] Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård Revisited: How to Construct a Hash Function. In: *Advances in Cryptology - CRYPTO '05*. Volume 3621 of *Lecture Notes in Computer Science*, Springer (2005) 21–39.
- [11] I. Damgård. Collision-free hash functions and public key signature schemes. *Advances in Cryptology - EUROCRYPT '87*. Volume 304 of *Lecture Notes in Computer Science*, Springer (1982) 416–427.
- [12] S. Halevi. EME*: Extending EME to handle arbitrary-length messages with associated data. *Advances in Cryptology – INDOCRYPT 2004*, LNCS vol. 3348, Springer, pp. 315–327, 2004.
- [13] S. Halevi and P. Rogaway. A parallelizable enciphering mode. *Topics in Cryptology – CT-RSA 2004*, LNCS vol. 2964, Springer, pp. 292–304, 2004.
- [14] S. Halevi and P. Rogaway. A tweakable enciphering mode. *Advances in Cryptology – CRYPTO 2003*, LNCS vol. 2729, Springer, pp. 482–499, 2003.
- [15] V. Lyubashevsky, D. Micciancio, C. Peikert, A. Rosen. Provably secure FFT hashing. *NIST 2nd Cryptographic Hash Function Workshop* (2006).
- [16] 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.
- [17] National Institute of Standards and Technology: FIPS PUB 180-1: Secure Hash Standard. (1995) Supersedes FIPS PUB 180 1993 May 11.
- [18] D. Pointcheval. The Composite Discrete Logarithm and Secure Authentication. *Third International Workshop on Practice and Theory in Public Key Cryptography - PKC '00*. Volume 1751 of *Lecture Notes in Computer Science*, Springer (2000) 113–128.
- [19] B. Preneel, R. Govaerts, J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. *Advances in Cryptology - CRYPTO 93*. Volume 773 of *Lecture Notes in Computer Science*, Springer (1994) 368–378.
- [20] M. Rabin. Digital signatures. *Foundations of secure computation*, R. A. Millo et. al. eds, Academic Press, 1978.
- [21] M. Rabin. Digital signatures and public key functions as intractable as factorization. *MIT Laboratory for Computer Science Report TR-212*, January 1979.
- [22] T. Ristenpart, P. Rogaway. How to Enrich the Message Space of a Cipher. To appear in *Fast Software Encryption - FSE '07*.
- [23] R. Rivest, A. Shamir, L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM* **21**(2): 120–126 (1978).
- [24] R. Rivest, Y. Tauman. Improved online/offline signature schemes. *Advances in Cryptology - CRYPTO '01*. Volume 2139 of *Lecture Notes in Computer Science*, Springer (2001) 355–367.
- [25] RSA Laboratories: RSA PKCS #1 v2.1: RSA Cryptography Standards (2002).
- [26] M.J. Saarinen. Security of VSH in the Real World. *Progress in Cryptology - INDOCRYPT '06*. Volume 4329 of *Lecture Notes in Computer Science*, Springer (2006) 95–103
- [27] 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.

- [28] 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.
- [29] D. Whitfield, M. Hellman. Privacy and Authentication: An Introduction to Cryptography. Proceedings of the IEEE, 67 (1979), pp. 397427.

A Proof of Theorem 5.1

Proof: Without loss we focus on adversaries that only query PGen a single time and at the beginning of the game. We therefore omit explicit mention of this oracle and instead just simply give \mathcal{A} the generated function at the beginning of the game. Note also that \mathcal{A} is assumed to never ask pointless queries. Our proof utilizes games [4]. We first over-view the game sequence.

We start with two games: game R0 (Real 0) emulates the oracles $(\mathcal{E}, E, D, \mathcal{F})$ whereas game I0 (Ideal 0) emulates the oracles $(\mathcal{I}, \mathcal{S}_E, \mathcal{S}_D, \mathcal{S}_\mathcal{F})$. We notate game transitions by $G \rightarrow G'$. We go through the transitions $R0 \rightarrow R1 \rightarrow R2 \rightarrow R3$ in order to make it have structure more like I0. At the same time we go through the transitions $I0 \rightarrow I1$ to make I0 have structure more like R0. These two game chains meet up with $R3 \rightarrow G0$ and $I1 \rightarrow G1$ where G0 and G1 are identical-until-bad. Then we begin modifying G1 in order to move to a game which we can use to build a “some-point one-way function” adversary. This last is a generalization of the usual one-way function definition, and is at the core of the security of TE. We bound the setting of bad in terms of such an adversary. Finally we give Lemma A.1, which bounds the advantage of any “some-point one way function” adversary by the advantage of a related (normal) one-way function adversary.

► Game R0 is shown in Figure 5. The subroutine *TESub* implements \mathcal{E} . Tables E and F are used to lazily build the ideal cipher E and random function \mathcal{F} .

► (R0 \rightarrow R1) We move from R0 to R1, shown in Figure 5, by removing permutivity restrictions on E . This means E and its inverse are replaced by two random functions. Note that in R1 \mathcal{O}_3 needs only return random bits — pointless queries are not allowed and the construction never utilizes the inverse of E . This is a lossy reduction and a birthday bound analysis gives us that

$$\text{Adv}(\mathcal{A}^{R0}, \mathcal{A}^{R1}) \leq \frac{(q_1\mu + q_2 + q_3)^2}{2^n}.$$

► (R1 \rightarrow R2) In this transition, we modify how \mathcal{O}_4 queries are handled. Now, instead of sampling directly for \mathcal{F} , \mathcal{O}_4 determines its return value via *TESub*. Particularly \mathcal{O}_4 runs *TESub*. If the same message M was not already executed through *TESub* (via a query to \mathcal{O}_1), then a new random string T^j is sampled and the first block returned by *TESub* is $f(T^j)$. Otherwise the value $f(T^i)$ is returned, where T^i is (what would have been) the range point of \mathcal{F} defined during the previous query to \mathcal{O}_1 . Thus we see that this code change still implements \mathcal{F} as before, but we no longer need to maintain F. On the other hand, running *TESub* for \mathcal{O}_4 queries also defines several range other random variables in the table E. However this is still conservative. If the same message was previously queried to \mathcal{O}_1 then those points were already defined already. Otherwise, these points are defined, but the adversary \mathcal{A} learns nothing about them until a later query to \mathcal{O}_1 or \mathcal{O}_2 . Thus we’ve merely chosen them early. Thus we have that

$$\text{Adv}(\mathcal{A}^{R1}, \mathcal{A}^{R2}) = 0.$$

► (R2 \rightarrow R3) We conservatively modify oracle \mathcal{O}_2 — simply adding an extra consistency check. For any K, C pair for which $E_K(C)$ is already defined from executing *TESub* and such that K corresponds

to a value returned to the adversary via an \mathcal{O}_4 query, we simply return the appropriate value as shown. This value is equal to the one in \mathbf{E} , and therefore we have that

$$\mathbf{Adv}(\mathcal{A}^{\mathbf{R}2}, \mathcal{A}^{\mathbf{R}3}) = 0.$$

► Game I0 is shown in Figure 7, and it implements the oracles $(\mathcal{R}, \mathcal{S}_E, \mathcal{S}_D, \mathcal{S}_F)$.

► (I0 \rightarrow I1) We conservatively transition from I0 to I1. In I1 we sample for \mathcal{I} by running a subroutine $TESub$. The distributions are unchanged, and note that although I0 records random choices in \mathbf{E} , consistency is not maintained. The distribution of responses to \mathcal{O}_2 queries are therefore the same as in I0. We have that

$$\mathbf{Adv}(\mathcal{A}^{\mathbf{I}0}, \mathcal{A}^{\mathbf{I}1}) = 0.$$

► (R3 \rightarrow G0 and I1 \rightarrow G1) We add a flag **bad** to game R0 to get game G0 and to I1 to derive G1, Figure 8. Clearly both changes are conservative, and now we have that G0 and G1 are identical-until-**bad1** or **bad2**. Applying the fundamental lemma of game-playing [4] gives

$$\mathbf{Adv}(\mathcal{A}^{\mathbf{G}0}, \mathcal{A}^{\mathbf{G}1}) \leq \Pr[\mathcal{A}^{\mathbf{G}1} \text{ sets bad1}].$$

► (G1 \rightarrow G2) Game G2 modifies how the flag **bad1** is set. Instead of recording random variables defined in a table \mathbf{E} (as in G1), G2 records in a set \mathcal{T}_1 all the range points corresponding to values T^j utilized in $TESub$ or values K queried to \mathcal{O}_2 . These latter values are only the ones which do not equal a T^j value resulting from a previous \mathcal{O}_4 query. We have that

$$\Pr[\mathcal{A}^{\mathbf{G}1} \text{ sets bad1}] \leq \Pr[\mathcal{A}^{\mathbf{G}2} \text{ sets bad1}]$$

since whenever G2 would set **bad** due to a collision in the table \mathbf{E} , game G1 finds a point already in \mathcal{T}_1 . (Note that the reverse is not necessarily true — G2 might set **bad1** in cases where G1 would not.)

► (G2 \rightarrow G3) Game G3 adds an input *flag* to $TESub$ and uses it to differentiate between \mathcal{O}_1 and \mathcal{O}_4 queries. The two cases, however, are not treated any differently compared to G2 (the boxed statements are not included). We also add a set \mathcal{T}_p that tracks the values T^j chosen in $TESub$ for \mathcal{O}_1 queries. We have that

$$\Pr[\mathcal{A}^{\mathbf{G}2} \text{ sets bad1}] = \Pr[\mathcal{A}^{\mathbf{G}3} \text{ sets bad1}].$$

Lastly a flag **bad2** is set whenever a collision occurs in \mathcal{T}_p .

► (G3 \rightarrow G4) Game G3 and G4 are identical-until-**bad2**, and G4 simply restricts sampling to avoid collisions in the choice of T^j values for (new) \mathcal{O}_1 queries. By the i^{th} such selection, the size of \mathcal{T}_p is at most $i - 1$. Counting across all \mathcal{O}_1 queries we have that

$$\Pr[\mathcal{A}^{\mathbf{G}4} \text{ sets bad2}] \leq \sum_{i=1}^{q_1} \frac{i-1}{2^k} = \frac{q_1^2 - q_1}{2^{k+1}}$$

and applying the fundamental lemma of game-playing

$$\begin{aligned} \Pr[\mathcal{A}^{\mathbf{G}3} \text{ sets bad1}] &\leq \Pr[\mathcal{A}^{\mathbf{G}4} \text{ sets bad1}] + \Pr[\mathcal{A}^{\mathbf{G}4} \text{ sets bad2}] \\ &= \frac{q_1^2 - q_1}{2^{k+1}} + \Pr[\mathcal{A}^{\mathbf{G}4} \text{ sets bad2}]. \end{aligned}$$

► (G4 \rightarrow G5) We modify G4 conservatively to get to game G5: \mathcal{O}_4 queries are checked up front to see if the message was earlier queried to \mathcal{O}_1 . If so the previously chosen Φ^i point is inverted and the preimage is returned. Otherwise $TESub$ is executed as before.

$$\Pr[\mathcal{A}^{\mathbf{G}4} \text{ sets bad1}] = \Pr[\mathcal{A}^{\mathbf{G}5} \text{ sets bad1}]$$

► (Adversary \mathcal{B}) Game G5 represents an algorithm that can invert f on some point. More particularly, G5 can be straightforwardly converted into a “some-point one way function” adversary. We define the some-point-owf advantage of an adversary \mathcal{B} against a function $f: Dom \rightarrow Rng$ by

$$\mathbf{Adv}_{\mathbb{F}}^{\text{spowf}}(\mathcal{B}) = \Pr \left[(f, f^{-1}) \stackrel{\$}{\leftarrow} \mathbb{F}; X \stackrel{\$}{\leftarrow} \mathcal{B}^{\text{Pl,Inv}}(f) : \mathcal{B} \text{ wins} \right].$$

The image oracle Pl, when invoked (on no input), chooses a novel random point $X' \in Dom$, records it, and returns $f(X')$. (By novel, we mean that the oracle samples from Dom without replacement.) The inverse oracle Inv, on input Y , returns $f^{-1}(Y)$ if Y was previously returned by Pl and otherwise it returns \perp (i.e., \mathcal{B} only gets to invert points already returned by Pl). The adversary wins if it can find a preimage X such that Y was returned from Pl and Y was not queried to Inv.

We construct a spowf-adversary \mathcal{B} from G5 as shown in Figure 10. The adversary runs exactly like G5 except that it (1) utilizes Pl to help answer \mathcal{O}_1 queries (for messages not already queried to \mathcal{O}_4); (2) utilizes Inv to invert images of f returned by previous \mathcal{O}_1 queries, and (3) it uses a table \mathbf{P} to track image, preimage pairs of f discovered through the course of the game. By the structure of the spowf security game we have that

$$\Pr [\mathcal{A}^{\text{G5}} \text{ sets bad1}] = \mathbf{Adv}_{\mathbb{F}}^{\text{spowf}}(\mathcal{B}).$$

► (Adversary \mathcal{C}) Now we prove the following lemma, which in turn completes the proof of Theorem 5.1.

Lemma A.1 *Let \mathbb{F} be a permutation family and \mathcal{B} be a spowf-adversary running in time t and making at most (q_i, q_p) queries, these totalling μ bits in length. Then there exists an adversary \mathcal{C} such that*

$$\mathbf{Adv}_{\mathbb{F}}^{\text{spowf}}(\mathcal{B}) \leq q_i \mathbf{Adv}_{\mathbb{F}}^{\text{owf}}(\mathcal{C})$$

where \mathcal{C} runs in time $t' \leq t + c\mu$ for some absolute constant c .

Proof: Given an adversary \mathcal{B} we construct an adversary \mathcal{C} as follows. On input f and target image Y^* , \mathcal{C} chooses a value $r \in [1..q_1]$ uniformly and sets a counter c to zero. It then runs \mathcal{B} . When \mathcal{B} queries Pl, adversary \mathcal{C} increments c . If $c = r$ then \mathcal{C} returns to \mathcal{B} the target image Y^* . Otherwise it chooses a random point X , records it, and returns $f(X)$. When \mathcal{B} queries Inv(Y), \mathcal{C} checks that Y was returned from a previous query to Pl. If not, it returns \perp . If instead $Y = Y^*$ then \mathcal{C} aborts. Otherwise it returns the appropriate preimage (which was earlier recorded).

Note that \mathcal{B} 's environment, as simulated by \mathcal{C} , is exactly that of the spowf experiment unless \mathcal{C} aborts early. Informally, if the choice r is correct, then \mathcal{B} won't force \mathcal{C} to halt (since by the rules of the spowf game, \mathcal{B} can't query to Inv the image point it will invert). A standard argument rigorously gives this, yielding that

$$\mathbf{Adv}_{\mathbb{F}}^{\text{spowf}}(\mathcal{B}) \leq q_i \mathbf{Adv}_{\mathbb{F}}^{\text{owf}}(\mathcal{C}).$$

■

Finally we add up the resources utilized by \mathcal{C} . Here \mathcal{B} runs G5^A and so \mathcal{B} runs in time $t_{\mathcal{B}} \leq t + t_{\mathcal{S}} + (q_2 + q_4)\text{Time}_f + (q_1 + q_2 + q_4)\log(q_1 + q_2 + q_4) + c'\mu$ where t is the time to run \mathcal{A} ; $t_{\mathcal{S}}$ is the time to run \mathcal{S} (whose code still appears within \mathcal{B}); Time_f is the time to compute f on some point; and the last term is the time needed to maintain \mathcal{T}_1 and find elements within it; and finally c is an absolute constant and μ is the total number of bits queried by \mathcal{A} . Inserting $t_{\mathcal{B}}$ into the resources utilized by \mathcal{C} and redefining absolute constants appropriately gives us the resource bound of \mathcal{C} stated in the theorem. ■

<p>procedure Initialize: R0 $j \leftarrow 0$; \mathbf{F}, \mathbf{E} everywhere \perp; $(f, f^{-1}) \xleftarrow{\\$} \mathbb{F}$; Ret f</p> <p>procedure $\mathcal{O}_1(M)$ * Implements \mathcal{E} * $j \leftarrow j + 1$; Ret $TESub(M, j)$</p> <p>subroutine $TESub(M, j)$ $M^j \leftarrow M$; $M_1^j \dots M_m^j \xleftarrow{\\$} M^j$ If $\exists i < j$ s.t. $M^i = M^j$ then Ret $f(T^i) \parallel Y_1^i \parallel \dots \parallel Y_m^i$ $T^j \xleftarrow{\\$} \{0, 1\}^k$; If $\mathbf{F}[M^j] \neq \perp$ then $T^j \leftarrow \mathbf{F}(M^j)$ $\mathbf{F}[M^j] \leftarrow T^j$ for $i = 1$ to m do $P_i^j \xleftarrow{\\$} \{0, 1\}^n$ If $P_i^j \in \mathit{Rng}(\mathbf{E}_{T^j})$ then $P_i^j \xleftarrow{\\$} \overline{\mathit{Rng}(\mathbf{E}_{T^j})}$ If $\mathbf{E}_{T^j}(i) \neq \perp$ then $P_i^j \leftarrow \mathbf{E}_{T^j}(i)$ $\mathbf{E}_{T^j}(i) \leftarrow P_i^j$ $Y_i^j \leftarrow P_i^j \oplus M_i^j$ Ret $f(T^j) \parallel Y_1^j \parallel \dots \parallel Y_m^j$</p> <p>procedure $\mathcal{O}_2(K, C)$ * Implements E * $U \xleftarrow{\\$} \{0, 1\}^n$ If $U \in \mathit{Rng}(\mathbf{E}_K)$ then $U \xleftarrow{\\$} \overline{\mathit{Rng}(\mathbf{E}_K)}$ If $\mathbf{E}_K(C) \neq \perp$ then $U \leftarrow \mathbf{E}_K(C)$ Ret $\mathbf{E}_K(C) \leftarrow U$</p> <p>procedure $\mathcal{O}_3(K, Y)$ * Implements E^{-1} * $C \xleftarrow{\\$} \{0, 1\}^n$ If $C \in \mathit{Dom}(\mathbf{E}_K)$ then $C \xleftarrow{\\$} \overline{\mathit{Dom}(\mathbf{E}_K)}$ If $\exists C'$ s.t. $\mathbf{E}_K(C') = Y$ then $C \leftarrow C'$ Ret $\mathbf{E}_K(C) \leftarrow Y$</p> <p>procedure $\mathcal{O}_4(M)$ * Implements G * $j \leftarrow j + 1$; $T^j \xleftarrow{\\$} \{0, 1\}^n$ If $\mathbf{F}(M) \neq \perp$ then $T^j \leftarrow \mathbf{F}(M)$ Ret $\mathbf{F}(M) \leftarrow T^j$</p>	
---	--

<p>procedure Initialize: R1 $j \leftarrow 0$; \mathbf{F}, \mathbf{E} everywhere \perp; $(f, f^{-1}) \xleftarrow{\\$} \mathbb{F}$; Ret f</p> <p>procedure $\mathcal{O}_1(M)$ $j \leftarrow j + 1$; Ret $TESub(M, j)$</p> <p>subroutine $TESub(M, j)$ $M^j \leftarrow M$; $M_1^j \dots M_m^j \xleftarrow{\\$} M^j$ If $\exists i < j$ s.t. $M^i = M^j$ then Ret $f(T^i) \parallel Y_1^i \parallel \dots \parallel Y_m^i$ $T^j \xleftarrow{\\$} \{0, 1\}^k$; If $\mathbf{F}[M^j] \neq \perp$ then $T^j \leftarrow \mathbf{F}(M^j)$ $\mathbf{F}[M^j] \leftarrow T^j$ for $i = 1$ to m do $P_i^j \xleftarrow{\\$} \{0, 1\}^n$ If $\mathbf{E}_{T^j}(i) \neq \perp$ then $P_i^j \leftarrow \mathbf{E}_{T^j}(i)$ $\mathbf{E}_{T^j}(i) \leftarrow P_i^j$ $Y_i^j \leftarrow P_i^j \oplus M_i^j$ Ret $f(T^j) \parallel Y_1^j \parallel \dots \parallel Y_m^j$</p> <p>procedure $\mathcal{O}_2(K, C)$ $U \xleftarrow{\\$} \{0, 1\}^n$ * remove permutivity * If $\mathbf{E}_K(C) \neq \perp$ then $U \leftarrow \mathbf{E}_K(C)$ Ret $\mathbf{E}_K(C) \leftarrow U$</p> <p>procedure $\mathcal{O}_3(K, C)$ Ret $D \xleftarrow{\\$} \{0, 1\}^n$ * remove permutivity *</p> <p>procedure $\mathcal{O}_4(M)$ $j \leftarrow j + 1$; $T^j \xleftarrow{\\$} \{0, 1\}^n$ If $\mathbf{F}(M) \neq \perp$ then $T^j \leftarrow \mathbf{F}(M)$ Ret $\mathbf{F}(M) \leftarrow T^j$</p>	
---	--

Figure 5: Game R0 implementing the oracles $(\mathcal{H}, E, D, \mathcal{F})$ and game R1, which replaces E and D by random functions.

procedure Initialize: R2
 $j \leftarrow 0$; \mathbf{F}, \mathbf{E} everywhere \perp ; $(f, f^{-1}) \stackrel{\$}{\leftarrow} \mathbb{F}$; Ret f

procedure $\mathcal{O}_1(M)$
 $j \leftarrow j + 1$; Ret $TESub(M, j)$

subroutine $TESub(M, j)$
 $M^j \leftarrow M$; $M_1^j \dots M_m^j \stackrel{n}{\leftarrow} M^j$
 If $\exists i < j$ s.t. $M^i = M^j$ then
 Ret $f(T^i) \parallel Y_1^i \parallel \dots \parallel Y_m^i$
 $T^j \stackrel{\$}{\leftarrow} \{0, 1\}^k$ * remove consistency check *
 for $i = 1$ to m do
 $P_i^j \stackrel{\$}{\leftarrow} \{0, 1\}^n$
 If $\mathbf{E}_{T^j}(i) \neq \perp$ then $P_i^j \leftarrow \mathbf{E}_{T^j}(i)$
 $\mathbf{E}_{T^j}(i) \leftarrow P_i^j$
 $Y_i^j \leftarrow P_i^j \oplus M_i^j$
 Ret $f(T^j) \parallel Y_1^j \parallel \dots \parallel Y_m^j$

procedure $\mathcal{O}_2(K, C)$
 $U \stackrel{\$}{\leftarrow} \{0, 1\}^n$
 If $\mathbf{E}_K(C) \neq \perp$ then $U \leftarrow \mathbf{E}_K(C)$
 Ret $\mathbf{E}_K(C) \leftarrow U$

procedure $\mathcal{O}_3(K, C)$
 Ret $D \stackrel{\$}{\leftarrow} \{0, 1\}^n$

procedure $\mathcal{O}_4(M)$
 $j \leftarrow j + 1$
 $Y_0 \parallel \tilde{Y} \leftarrow TESub(M, j)$ * sample using $TESub$ *
 Ret $\Gamma^j \leftarrow f^{-1}(Y_0)$

procedure Initialize: R3
 $j \leftarrow 0$; \mathbf{F}, \mathbf{E} everywhere \perp ; $(f, f^{-1}) \stackrel{\$}{\leftarrow} \mathbb{F}$; Ret f

procedure $\mathcal{O}_1(M)$
 $j \leftarrow j + 1$; Ret $TESub(M, j)$

subroutine $TESub(M, j)$
 $M^j \leftarrow M$; $M_1^j \dots M_m^j \stackrel{n}{\leftarrow} M^j$
 If $\exists i < j$ s.t. $M^i = M^j$ then
 Ret $f(T^i) \parallel Y_1^i \parallel \dots \parallel Y_m^i$
 $T^j \stackrel{\$}{\leftarrow} \{0, 1\}^k$
 for $i = 1$ to m do
 $P_i^j \stackrel{\$}{\leftarrow} \{0, 1\}^n$
 If $\mathbf{E}_{T^j}(i) \neq \perp$ then $P_i^j \leftarrow \mathbf{E}_{T^j}(i)$
 $\mathbf{E}_{T^j}(i) \leftarrow P_i^j$
 $Y_i^j \leftarrow P_i^j \oplus M_i^j$
 Ret $f(T^j) \parallel Y_1^j \parallel \dots \parallel Y_m^j$

procedure $\mathcal{O}_2(K, C)$
 If $\exists i$ s.t. $\Gamma^i = K$ and $C \leq |M^i|/n$ then
 Ret $M_C^i \oplus Y_C^i$ * extra consistency check *
 $U \stackrel{\$}{\leftarrow} \{0, 1\}^n$
 If $\mathbf{E}_K(C) \neq \perp$ then $U \leftarrow \mathbf{E}_K(C)$
 Ret $\mathbf{E}_K(C) \leftarrow U$

procedure $\mathcal{O}_3(K, C)$
 Ret $D \stackrel{\$}{\leftarrow} \{0, 1\}^n$

procedure $\mathcal{O}_4(M)$
 $j \leftarrow j + 1$
 $Y_0 \parallel \tilde{Y} \leftarrow TESub(M, j)$
 Ret $\Gamma^j \leftarrow f^{-1}(Y_0)$

Figure 6: Game R2 and game R3 encapsulate conservative modifications which move the real games closer to the ideal games.

procedure Initialize: I0
 $j \leftarrow 0$; \mathbf{I} everywhere \perp ; $(f, f^{-1}) \stackrel{\$}{\leftarrow} \mathbb{F}$; Ret f

procedure $\mathcal{O}_1(M)$ * Implements \mathcal{I} *
 $Y_0 \parallel \tilde{Y} \stackrel{\$}{\leftarrow} \{0, 1\}^{|M|+\tau}$
 If $\mathbf{I}(M) \neq \perp$ then $Y_0 \parallel \tilde{Y} \leftarrow \mathbf{I}(M)$
 Ret $\mathbf{I}(M) \leftarrow Y_0 \parallel \tilde{Y}$

procedure $\mathcal{O}_2(K, C)$ * Implements \mathcal{S}_E *
 If $\exists i$ s.t. $\Gamma^i = K$ and $C \leq |M^i|/n$ then
 Ret $M_C^i \oplus Y_C^i$
 Ret $U \stackrel{\$}{\leftarrow} \{0, 1\}^n$

procedure $\mathcal{O}_3(K, Y)$ * Implements \mathcal{S}_D *
 Return $D \stackrel{\$}{\leftarrow} \{0, 1\}^n$

procedure $\mathcal{O}_4(M)$ * Implements \mathcal{S}_F *
 $j \leftarrow j + 1$; $M_1^j \cdots M_m^j \stackrel{\$}{\leftarrow} M$
 $Y_0^j \parallel \tilde{Y}^j \stackrel{\$}{\leftarrow} \{0, 1\}^{|M^j|+\tau}$
 If $\mathbf{I}(M^j) \neq \perp$ then $Y_0^j \parallel \tilde{Y}^j \leftarrow \mathbf{I}(M^j)$
 $\mathbf{I}(M^j) \leftarrow Y_0^j \parallel \tilde{Y}^j$
 Ret $\Gamma^j \leftarrow f^{-1}(Y_0^j)$

procedure Initialize: I1
 $j \leftarrow 0$; \mathbf{I}, \mathbf{E} everywhere \perp ; $(f, f^{-1}) \stackrel{\$}{\leftarrow} \mathbb{F}$; Ret f

procedure $\mathcal{O}_1(M)$
 $j \leftarrow j + 1$; Ret $TESub(M, j)$ * use $TESub$ to sample *

subroutine $TESub(M, j)$
 $M^j \leftarrow M$; $M_1^j \cdots M_m^j \stackrel{\$}{\leftarrow} M^j$
 If $\exists i < j$ s.t. $M^i = M^j$ then
 Ret $f(T^i) \parallel Y_1^i \parallel \cdots \parallel Y_m^i$
 $T^j \stackrel{\$}{\leftarrow} \{0, 1\}^k$
 for $i = 1$ to m do
 $P_i^j \stackrel{\$}{\leftarrow} \{0, 1\}^n$
 $\mathbf{E}_{T^j}(i) \leftarrow P_i^j$
 $Y_i^j \leftarrow P_i^j \oplus M_i^j$
 Ret $f(T^j) \parallel Y_1^j \parallel \cdots \parallel Y_m^j$

procedure $\mathcal{O}_2(K, C)$
 If $\exists i$ s.t. $\Gamma^i = K$ and $C \leq |M^i|/n$ then
 Ret $M_C^i \oplus Y_C^i$
 Ret $\mathbf{E}_K(C) \stackrel{\$}{\leftarrow} \{0, 1\}^n$ * record random choice *

procedure $\mathcal{O}_3(K, Y)$
 Return $D \stackrel{\$}{\leftarrow} \{0, 1\}^n$

procedure $\mathcal{O}_4(M)$
 $j \leftarrow j + 1$
 $Y_0^j \parallel \tilde{Y}^j \leftarrow TESub(M, j)$ * use $TESub$ to sample *
 Ret $\Gamma^j \leftarrow f^{-1}(Y_0^j)$

Figure 7: Games I0 and I1 (ideal 0 and 1). The former implements the oracles $(\mathcal{I}, \mathcal{S}_E, \mathcal{S}_D, \mathcal{S}_F)$ and the latter is a conservative change to closer resemble to R3.

<p>procedure Initialize: G0 G1</p> <p>$j \leftarrow 0$; E everywhere \perp; $(f, f^{-1}) \stackrel{\\$}{\leftarrow} \mathbb{F}$; Ret f</p> <p>procedure $\mathcal{O}_1(M)$</p> <p>$j \leftarrow j + 1$; Ret $TESub(M, j)$</p> <p>subroutine $TESub(M, j)$</p> <p>$M^j \leftarrow M$; $M_1^j \dots M_m^j \stackrel{\\$}{\leftarrow} M^j$</p> <p>If $\exists i < j$ s.t. $M^i = M^j$ then</p> <p style="padding-left: 20px;">Ret $f(T^i) \parallel Y_1^i \parallel \dots \parallel Y_m^i$</p> <p>$T^j \stackrel{\\$}{\leftarrow} \{0, 1\}^k$</p> <p>for $i = 1$ to m do</p> <p style="padding-left: 20px;">$P_i^j \stackrel{\\$}{\leftarrow} \{0, 1\}^n$</p> <p style="padding-left: 20px;">If $E_{T^j}(i) \neq \perp$ then</p> <p style="padding-left: 40px;">bad1 \leftarrow true * add flag bad *</p> <p style="padding-left: 40px;">$P_i^j \leftarrow E_{T^j}(i)$</p> <p style="padding-left: 20px;">$E_{T^j}(i) \leftarrow P_i^j$</p> <p style="padding-left: 20px;">$Y_i^j \leftarrow P_i^j \oplus M_i^j$</p> <p>Ret $f(T^j) \parallel Y_1^j \parallel \dots \parallel Y_m^j$</p> <p>procedure $\mathcal{O}_2(K, C)$</p> <p>If $\exists i$ s.t. $\Gamma^i = K$ and $C \leq M^i /n$ then</p> <p style="padding-left: 20px;">Ret $M_C^i \oplus Y_C^i$</p> <p>$U \stackrel{\\$}{\leftarrow} \{0, 1\}^n$</p> <p>If $E_K(C) \neq \perp$ then</p> <p style="padding-left: 20px;">bad1 \leftarrow true * add flag bad *</p> <p style="padding-left: 20px;">$U \leftarrow E_K(C)$ Ret $E_K(C) \stackrel{\\$}{\leftarrow} \{0, 1\}^n$</p> <p>procedure $\mathcal{O}_3(K, Y)$</p> <p>Return $D \stackrel{\\$}{\leftarrow} \{0, 1\}^n$</p> <p>procedure $\mathcal{O}_4(M)$</p> <p>$j \leftarrow j + 1$</p> <p>$Y_0^j \parallel \tilde{Y}^j \leftarrow TESub(M, j)$</p> <p>Ret $\Gamma^j \leftarrow f^{-1}(Y_0^j)$</p>	<p>G1</p>
---	-----------

<p>procedure Initialize: G2</p> <p>$j \leftarrow 0$; $(f, f^{-1}) \stackrel{\\$}{\leftarrow} \mathbb{F}$; $\mathcal{T}_1 \leftarrow \emptyset$; Ret f</p> <p style="text-align: center;">* remove table E; add set \mathcal{T}_1 *</p> <p>procedure $\mathcal{O}_1(M)$</p> <p>$j \leftarrow j + 1$; Ret $TESub(M, j)$</p> <p>subroutine $TESub(M, j)$</p> <p>$M^j \leftarrow M$; $M_1^j \dots M_m^j \stackrel{\\$}{\leftarrow} M^j$</p> <p>If $\exists i < j$ s.t. $M^i = M^j$ then</p> <p style="padding-left: 20px;">Ret $\Phi^i \parallel Y_1^i \parallel \dots \parallel Y_m^i$</p> <p>$T^j \stackrel{\\$}{\leftarrow} \{0, 1\}^k$; $\Phi^j \leftarrow f(T^j)$ * compute range point early *</p> <p>If $\Phi^j \in \mathcal{T}_1$ then bad1 \leftarrow true * set bad1 aggressively *</p> <p>for $i = 1$ to m do</p> <p style="padding-left: 20px;">$P_i^j \stackrel{\\$}{\leftarrow} \{0, 1\}^n$</p> <p style="padding-left: 20px;">$Y_i^j \leftarrow P_i^j \oplus M_i^j$</p> <p>$\mathcal{T}_1 \stackrel{\cup}{\leftarrow} \Phi^j$ * record Φ^j *</p> <p>Ret $\Phi^j \parallel Y_1^j \parallel \dots \parallel Y_m^j$</p> <p>procedure $\mathcal{O}_2(K, C)$</p> <p>If $\exists i$ s.t. $\Gamma^i = K$ and $C \leq M^i /n$ then</p> <p style="padding-left: 20px;">Ret $M_C^i \oplus Y_C^i$</p> <p>$\Phi \leftarrow f(K)$ * compute range point *</p> <p>If $\Phi \in \mathcal{T}_1$ then bad1 \leftarrow true * set bad1 aggressively *</p> <p>$\mathcal{T}_1 \stackrel{\cup}{\leftarrow} \Phi$ * add $f(K)$ to image set *</p> <p>Ret $U \stackrel{\\$}{\leftarrow} \{0, 1\}^n$ * return random point *</p> <p>procedure $\mathcal{O}_3(K, Y)$</p> <p>Return $D \stackrel{\\$}{\leftarrow} \{0, 1\}^n$</p> <p>procedure $\mathcal{O}_4(M)$</p> <p>$j \leftarrow j + 1$</p> <p>$Y_0^j \parallel \tilde{Y}^j \leftarrow TESub(M, j)$</p> <p>Ret $\Gamma^j \leftarrow f^{-1}(Y_0^j)$</p>	<p>G2</p>
---	-----------

Figure 8: Games G0 (boxed statements included) is equivalent to R3. Game G1 (boxed statements not included) is equivalent to I1. Game G2 is a modification of G1 that dispenses with E and more aggressively sets bad1.

procedure Initialize: $j \leftarrow 0; (f, f^{-1}) \xleftarrow{\$} \mathbb{F}; \mathcal{T}_p \leftarrow \mathcal{T}_1 \leftarrow \emptyset; \text{Ret } f$ * add set \mathcal{T}_p *	G3 G4
procedure $\mathcal{O}_1(M)$ $j \leftarrow j + 1; \text{Ret } \text{TESub}(M, j)$	
subroutine $\text{TESub}(M, j, \text{flag})$ * add flag * $M^j \leftarrow M; M_1^j \dots M_m^j \xleftarrow{\$} M^j$ If $\exists i < j$ s.t. $M^i = M^j$ then Ret $\Phi^i \parallel Y_1^i \parallel \dots \parallel Y_m^i$ If $\text{flag} = 1$ then * split into two cases * $T^j \xleftarrow{\$} \{0, 1\}^k$ If $T^j \in \mathcal{T}_p$ then bad2 \leftarrow true $T^j \xleftarrow{\\$} \{0, 1\}^k \setminus \mathcal{T}_p$ * G3: add flag; G4: restrict sampling * $\mathcal{T}_p \xleftarrow{\cup} T^j; \Phi^j \leftarrow f(T^j)$ else $T^j \xleftarrow{\$} \{0, 1\}^k; \Phi^j \leftarrow f(T^j)$ If $\Phi^j \in \mathcal{T}_1$ then bad1 \leftarrow true for $i = 1$ to m do $P_i^j \xleftarrow{\$} \{0, 1\}^n$ $Y_i^j \leftarrow P_i^j \oplus M_i^j$ $\mathcal{T}_1 \xleftarrow{\cup} \Phi^j$ Ret $\Phi^j \parallel Y_1^j \parallel \dots \parallel Y_m^j$	
procedure $\mathcal{O}_2(K, C)$ If $\exists i$ s.t. $\Gamma^i = K$ and $C \leq M^i /n$ then Ret $M_C^i \oplus Y_C^i$ $\Phi \leftarrow f(K)$ If $\Phi \in \mathcal{T}_1$ then bad1 \leftarrow true $\mathcal{T}_1 \xleftarrow{\cup} \Phi$ Ret $U \xleftarrow{\$} \{0, 1\}^n$	
procedure $\mathcal{O}_3(K, Y)$ Return $D \xleftarrow{\$} \{0, 1\}^n$	
procedure $\mathcal{O}_4(M)$ $j \leftarrow j + 1$ $Y_0^j \parallel \tilde{Y}^j \leftarrow \text{TESub}(M, j)$ Ret $\Gamma^j \leftarrow f^{-1}(Y_0^j)$	

procedure Initialize: $j \leftarrow 0; (f, f^{-1}) \xleftarrow{\$} \mathbb{F}; \mathcal{T}_p \leftarrow \mathcal{T}_1 \leftarrow \emptyset; \text{Ret } f$	G5
procedure $\mathcal{O}_1(M)$ $j \leftarrow j + 1; \text{Ret } \text{TESub}(M, j, 1)$	
subroutine $\text{TESub}(M, j, \text{flag})$ $M^j \leftarrow M; M_1^j \dots M_m^j \xleftarrow{\$} M^j$ If $\exists i < j$ s.t. $M^i = M^j$ then Ret $\Phi^i \parallel Y_1^i \parallel \dots \parallel Y_m^i$ If $\text{flag} = 1$ then $T^j \xleftarrow{\$} \{0, 1\}^k \setminus \mathcal{T}_p; \mathcal{T}_p \xleftarrow{\cup} T^j$ * simplify code * $\Phi^j \leftarrow f(T^j)$ else $T^j \xleftarrow{\$} \{0, 1\}^k; \Phi^j \leftarrow f(T^j)$ If $\Phi^j \in \mathcal{T}_1$ then bad1 \leftarrow true for $i = 1$ to m do $P_i^j \xleftarrow{\$} \{0, 1\}^n$ $Y_i^j \leftarrow P_i^j \oplus M_i^j$ $\mathcal{T}_1 \xleftarrow{\cup} \Phi^j$ Ret $\Phi^j \parallel Y_1^j \parallel \dots \parallel Y_m^j$	
procedure $\mathcal{O}_2(K, C)$ If $\exists i$ s.t. $\Gamma^i = K$ and $C \leq M^i /n$ then Ret $M_C^i \oplus Y_C^i$ $\Phi \leftarrow f(K)$ If $\Phi \in \mathcal{T}_1$ then bad1 \leftarrow true $\mathcal{T}_1 \xleftarrow{\cup} \Phi$ Ret $U \xleftarrow{\$} \{0, 1\}^n$	
procedure $\mathcal{O}_3(K, Y)$ Return $D \xleftarrow{\$} \{0, 1\}^n$	
procedure $\mathcal{O}_4(M)$ $j \leftarrow j + 1$ If $\exists i$ s.t. $M^i = M$ then Ret $\Gamma^j \leftarrow f^{-1}(\Phi^i)$ * add check for earlier \mathcal{O}_1 query * $Y_0^j \parallel \tilde{Y}^j \leftarrow \text{TESub}(M, j, 4)$ Ret $\Gamma^j \leftarrow T^j$	

Figure 9: Games G3 modifies how T^j values are selected and introduces a new flag **bad2**. Game G4 adds code to restrict the sampling of T^j so that collisions do not occur. They are identical-until-**bad2**. Game G5 is a conservative change from G4, where some of the code is modified in preparation for building a “some-point” one-way function adversary.

Adversary $\mathcal{B}^{\text{Pl,Inv}}(f)$
 $j \leftarrow 0$; $\mathcal{T}_1 \leftarrow \emptyset$; P everywhere undefined
Run $\mathcal{A}(f)$ answering oracle queries as follows:

Query $\mathcal{O}_1(M)$
 $j \leftarrow j + 1$; Ret $TESub(M, j)$

subroutine $TESub(M, j, flag)$
 $M^j \leftarrow M$; $M_1^j \dots M_m^j \stackrel{\$}{\leftarrow} M^j$
If $\exists i < j$ s.t. $M^i = M^j$ then
Ret $\Phi^i \parallel Y_1^i \parallel \dots \parallel Y_m^i$
If $flag = 1$ then
 $\Phi^j \leftarrow \text{Pl}$
else
 $T^j \stackrel{\$}{\leftarrow} \{0, 1\}^k$; $\Phi^j \leftarrow f(T^j)$; $\mathsf{P}[\Phi^j] \leftarrow T^j$
If $\Phi^j \in \mathcal{T}_i$ then $\text{bad1} \leftarrow \text{true}$; Output $\mathsf{P}[\Phi^j]$
for $i = 1$ to m do
 $P_i^j \stackrel{\$}{\leftarrow} \{0, 1\}^n$
 $Y_i^j \leftarrow P_i^j \oplus M_i^j$
 $\mathcal{T}_1 \stackrel{\leftarrow}{\cup} \Phi^j$
Ret $\Phi^j \parallel Y_1^j \parallel \dots \parallel Y_m^j$

Query $\mathcal{O}_2(K, C)$
If $\exists i$ s.t. $\Gamma^i = K$ and $C \leq |M^i|/n$ then
Ret $M_C^i \oplus Y_C^i$
 $\Phi \leftarrow f(K)$
If $\Phi \in \mathcal{T}_i$ then $\text{bad1} \leftarrow \text{true}$; Output K
 $\mathcal{T}_1 \stackrel{\leftarrow}{\cup} \Phi$; $\mathsf{P}[\Phi] \leftarrow K$
Ret $U \stackrel{\$}{\leftarrow} \{0, 1\}^n$

Query $\mathcal{O}_3(K, Y)$
Return $D \stackrel{\$}{\leftarrow} \{0, 1\}^n$

procedure $\mathcal{O}_4(M)$
 $j \leftarrow j + 1$
If $\exists i$ s.t. $M^i = M$ then Ret $\Gamma^j \leftarrow \text{Inv}(\Phi^i)$
 $Y_0^j \parallel \tilde{Y}^j \leftarrow TESub(M, j, 4)$
Ret $\Gamma^j \leftarrow T^j$

If \mathcal{A} halts, then output \perp

Figure 10: The some-point one-way function adversary \mathcal{B} , which is essentially $\text{G5}^{\mathcal{A}}$.