A preliminary version of this paper appears in *Advances in Cryptology – ASIACRYPT '07*, Lecture Notes in Computer Science Vol. 4833, pp. 147–163, K. Kurosawa ed., Springer-Verlag, 2007. This is the full version.

How to Build a Hash Function from any Collision-Resistant Function

Thomas Ristenpart* Thomas Shrimpton†‡

April 2008

Abstract

Recent collision-finding attacks against hash functions such as MD5 and SHA-1 motivate the use of *provably* collision-resistant (CR) functions in their place. Finding a collision in a provably CR function implies the ability to solve some hard problem (e.g., factoring). Unfortunately, existing provably CR functions make poor replacements for hash functions as they fail to deliver behaviors demanded by practical use. In particular, they are easily distinguished from a random oracle. We initiate an investigation into building hash functions from provably CR functions. As a method for achieving this, we present the Mix-Compress-Mix (MCM) construction; it envelopes any provably CR function H (with suitable regularity properties) between two injective "mixing" stages. The MCM construction simultaneously enjoys (1) provable collision-resistance in the standard model, and (2) indifferentiability from a monolithic random oracle when the mixing stages themselves are indifferentiable from a random oracle that observes injectivity. We instantiate our new design approach by specifying a blockcipher-based construction that appropriately realizes the mixing stages.

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

*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 Mihir Bellare's NSF grant CNS 0524765 and his gift from Intel Corporation.

†Dept. of Computer Science, Portland State University Room 120, Forth Avenue Building, 1900 SW 4th Avenue, Portalnd OR 97201, USA. Email: teshrim@cs.pdx.edu. URL: http://www.cs.pdx.edu/. Supported by NSF grant CNS 0627752.

‡ Faculty of Informatics, University of Lugano Via Buffi 13, CH-6900 Lugano, Switzerland. Email: thomas.shrimpton@unisi.ch. URL: http://www.inf.unisi.ch/faculty/shrimpton/.

Contents

1 Introduction

BACKGROUND. SHA-1, a Merkle-Damgård style [24, 15] iterated function, is provably collision resistant under the *assumption* that its underlying compression function is collision resistant. But the recent collision-finding attacks against SHA-1 (and related hash functions) [37, 38] have made clear the point that assumptions of collision resistance are often unfounded in practice.

Rather than assuming collision resistance outright, several works [12, 22, 26, 33, 14] build functions for which the guarantee of collision resistance rests, in a provable way, on the hardness of some well-studied computational problem. As a simple example, consider the function $H(m) = x^m \bmod n$ where x is some fixed base and n is a (supposedly) hard-to-factor composite [26, 33]. This function is (what we shall call) *provably* CR since there exists a formal reduction showing that the ability to find collisions in H implies the ability to efficiently factor n.

But such a collision-resistant function is not a *hash function*, at least not when one attempts to define a hash function by its myriad uses in practice[1]. For example, hash functions are frequently used as a way to compress and 'mix-up' strings of bits in an 'unpredictable' way; here it seems clear that the intent is for the hash function to mimic a random oracle, a publicly available random function with a large domain. Unfortunately, the provably CR function H is a poor real-world instantiation of a random oracle. Note, for example, that $H(2m) \equiv H(m)^2 \pmod n$, which would be true with exceedingly small probability if H were instead a random oracle. The very structure that gives H and other provably CR functions their collision-resistance thus renders them useless for many practical applications of hash functions [12, 36].

On the other hand, recent results [13, 2, 11] offer constructions that 'behave' as random oracles (and are called pseudorandom oracles, or PROs) when the underlying primitives are themselves idealized objects, like fixed-input length random oracles or ideal ciphers. In theory then, a PRO is a secure hash functions in a very broad sense. But the security guarantees offered by a PRO only hold in an idealized model. When one steps outside of the ideal model in which the security proofs take place, the actual security guarantees are much less clear. As an example, Bellare and Ristenpart [2] have pointed out that the PRO constructions from [13] fail to be collision resistant when the underlying compression function is only assumed to be CR (rather than being a fixed-input-length random oracle).

THIS PAPER. We begin an investigation into methods for building functions that are *both* provably CR in the standard model and provably pseudorandom oracles in idealized models. In particular, we offer a generic construction that we call Mix-Compress-Mix, or MCM; See Figure 1. Essentially MCM is a way to encapsulate a provably CR function in such a way that the resulting object is a PRO when the encapsulation steps behave ideally, and yet remains provably collision resistant in the standard model (i.e., when the encapsulation steps are only complexity theoretic objects).

The construction is simple: first apply an injective "mixing" step $\mathcal{E}_1$ to the input message, then compress the result using a provably CR function H, and finally apply a second injective "mixing" step $\mathcal{E}_2$ to produce the output. Here H and $\mathcal{E}_1$ can accept variable-input-lengths. Note that since MCM is building a hash function, the mixing steps $\mathcal{E}_1$ and $\mathcal{E}_2$ are necessarily deterministic and publically computable functions. By demanding that they also be injective, we have immediately that collisions against MCM imply collisions against H. We stress that *no* cryptographic assumptions about the mixing steps are needed to prove collision resistance of MCM.

At the same time, MCM behaves like a random oracle when $\mathcal{E}_1, \mathcal{E}_2$ are PROs, and the CR hash function is close to regular (i.e., the preimage set of any particular output isn't too large). In

[1] This viewpoint is not ours alone. One of the designers of VSH [12], Arjen Lenstra, once publicly stated "VSH is not a hash function."

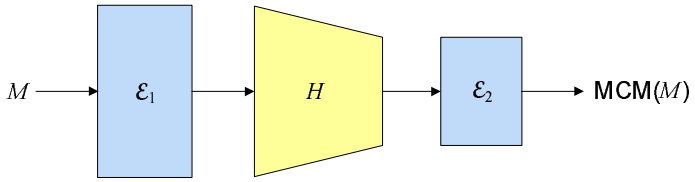

Figure 1: The MCM construction: H is a collision resistant hash function, and $\mathcal{E}_1, \mathcal{E}_2$ are mixing functions. All three components of MCM must be deterministic and publically computable.

fact, we will actually construct $\mathcal{E}_1, \mathcal{E}_2$ to be *pseudorandom injective oracles*, or PRIOs; we'll say more about these in a moment. To make precise our use of the word "behaves" above, we use the indifferentiability framework of Maurer et al. [23]. We'll prove that MCM is indifferentiable from a monolithic random oracle when the mixing steps $\mathcal{E}_1$ and $\mathcal{E}_2$ are indifferentiable from random oracles (that observe injectivity). While the formal results are quite technical, the practical intuition behind the security of MCM is straightforward: the mixing steps obfuscate input-output relationships of the underlying compressing step. Recall our provably CR example $H(m) = x^m \bmod n$ and the associated attack that distinguished it from a random oracle. 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 $\mathcal{E}_2$, too.

One might be tempted to think a construction even simpler than MCM meets our goals. In Section 4 we discuss natural simplifications of MCM (e.g., dropping $\mathcal{E}_1$ or lifting our stringent injectivity requirements), showing that these fall short in one way or another. Moreover, we review in more detail why existing approaches for building hash functions also fail.

Although we have just described MCM in the variable-input-length setting, we note that it also works for building a dual-property compression function (i.e., a fixed-input-length function) from any CR compression function. The result could be then be used inside a multi-property-preserving domain extension transform such as EMD [2].

A new approach to hash function design. By generically composing appropriate mixing and compressing stages, MCM allows the following 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. This approach is a significant departure from traditional hash function designs, in which one typically constructs a compression function that must necessarily (and simultaneously) be secure in various ways. With MCM, we instead build a hash function by designing components to achieve specific security goals. The benefits of such specialized components are immediate: MCM allows building a single hash function that has very strong CR guarantees while simultaneously being suitable for instantiating a random oracle.

Secure mixing steps. Remaining is the question of how to build mixing steps sufficient for the goals of MCM. As we've said, we require the mixing steps to be both injective *and* indifferentiable from a random oracle that observes injectivity. At first glance these requirements might seem overly burdensome. Can't the requirement simply be for the mixing steps to realize pseudorandom oracles, which we already know (via [13, 11, 2]) how to build? No: while a PRO would satisfy the second constraint, albeit with some additive birthday-bound loss in concrete security, it would not

at the same time suffice for MCM's crucial standard-model CR guarantee. This is because a PRO provides no guarantees of collision-resistance outside of an idealized model. In fact, simultaneously satisfying both requirements, injectivity and indifferentiability, is technically challenging.

To our knowledge, building a PRIO has never been considered before. Dodis and Puniya [17, 16] consider a similar goal, that of building random permutations from random functions, but these are invertible by construction, whereas PRIOs are not. Moreover, their proofs of security only hold for honest-but-curious adversaries. We therefore present the Tag-and-Encipher (TE) construction for realizing a PRIO (see Section 6). It is a blockcipher mode of operation (which also employs a single trapdoor one-way permutation call) that is injective by construction. In the ideal cipher model and under the assumption of trusted setup of the trapdoor permutation, the TE construction is indifferentiable from an injective random oracle. While not particularly efficient, we view the TE construction as a proof-of-concept, and hope it fosters future efforts to build these novel primitives.

Notes on indifferentiability and composability. In order to accomplish our task of building a hash function with both strong standard model and ideal model guarantees, we exercise the indifferentiability framework in novel ways. First, both MCM and TE are a combination of complexity-theoretic objects (the CR function H and the trapdoor permutation) and information-theoretic objects (the idealized components). Previous indifferentiability results have been solely information-theoretic. Second, our model allows the simulator to choose the trapdoor permutation utilized in TE. These two facts imply limitations on the generic composability of our schemes. Composability refers to the guarantee that *any* cryptographic scheme proven secure using an ideal object remains secure when this object is replaced by a construction that is indifferentiable from it. In practice the limited composability of our constructions means that they might not be suitable for all applications of random (injective) oracles. We discuss this matter in more detail, and pose some interesting open questions raised by it, in Section 8.

2 Preliminaries

Notation. 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 $X = X[1]X[2]\cdots X[|X|]$. When x is an integer we write $\langle x \rangle_n$ to represent some canonical encoding of the integer as a bit string of n bits (necessarily $n \geq \lceil \log x \rceil$). Overloading our notation, when X is a bit string we write $\langle X \rangle_n$ to represent the integer x encoded by X. When clear from context we will omit the subscript n. 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 $S \xleftarrow{\cup} x$, which means $S \leftarrow S \cup \{x\}$. For sets of bit strings S and S' we write $S \,||\, S'$ to denote the set of strings $s \,||\, s'$ such that $s \in S$ and $s \in S'$. For an algorithm f, we define $\mathsf{Time}_f(\mu)$ as the worst-case time to compute f on a message of length at most μ.

For bit string M, we write $M_1 \cdots M_m \xleftarrow{n} M$ for the following procedure. Let $m = \lceil |M|/n \rceil$. Then assign to M_i the i^{th} n-bit string of M for each $1 \leq i < m$. If $|M| \bmod n = 0$ then assign M_m the last n bits of M. Otherwise assign to M_m the last $|M| \bmod n$ bits of M. Similarly for string Y such that $|Y| > k$, we write $Y_0 \cdots Y_m \xleftarrow{k,n} Y$ for the following procedure. Let $m = \lceil (|Y| - k)/n \rceil$. Then assign to Y_0 the first k bits of Y. Let Y' be the remaining $|Y| - k$ bits. Then compute $Y_1 \cdots Y_m \xleftarrow{n} Y'$.

Cryptographic functions. A *(keyed) cryptographic function* $F = (\mathcal{K}, F, Dom, Rng, \tau)$ is a pair of algorithms, two (non-empty) sets of bit strings, and an optional parameter (the function's stretch). The key generation algorithm $\mathcal{K}$ outputs a key. We abuse notation slightly and write $k \in \mathcal{K}$ if k is output by $\mathcal{K}$ for some set of random coins. Each $k \in \mathcal{K}$ specifies a function F_k: $Dom \to Rng$.

We interchangeably write $F_k(\cdot)$ or $F(k,\cdot)$. If $\tau \geq 0$ is specified, this is the constant stretch and it specifies that $|F_k(M)| = |M| + \tau$ for all $k \in \mathcal{K}$ and all $M \in \mathit{Dom}$. If the construction is keyless then $\mathcal{K}$ is omitted and F only takes input a message. We will write $(\mathcal{K}, F, \mathit{Dom}, r)$ if $\mathit{Rng} = \{0,1\}^r$ and $(\mathcal{K}, F, d, r)$ if additionally $\mathit{Dom} = \{0,1\}^d$. A function might utilize primitives $f_1, \ldots, f_l$ as black-boxes for some number $l \geq 0$. In this case we write $F^{f_1,\ldots,f_l}(k, M)$ to mean computing F on key K and message $M \in \mathit{Dom}$ using oracle access to $f_1, \ldots, f_l$. A cryptographic function is *injective* if $F(k, M) = F(k, M')$ implies that $M = M'$ for all $k \in \mathcal{K}$.

Collision resistance. Let $F = (\mathcal{K}, F, \mathit{Dom}, \mathit{Rng})$ be a cryptographic function. Then we define the collision-finding advantage of an adversary $\mathcal{A}$ against F as

$$\mathbf{Adv}^{\mathrm{cr}}_{f}(\mathcal{A}) = \Pr\left[F_k(X) = F_k(X') \wedge X \neq X' \ : \ k \xleftarrow{\$} \mathcal{K}; (X, X') \xleftarrow{\$} \mathcal{A}(k) \right]$$

where the probability is over the random coins used by $\mathcal{K}$ and $\mathcal{A}$.

Regularity. A function is *regular* if each image has an equal number of preimages. A cryptographic function $F = (\mathcal{K}, F, \mathit{Dom}, \mathit{Rng})$ is regular if, for all $k \in \mathcal{K}$, the map F_k: $\mathit{Dom} \to \mathit{Rng}$ is regular. Associated to F is the set $\mathsf{Preim}(k, \ell, Y) = \{X : X \in \mathit{Dom} \wedge |X| = \ell \wedge F_k(X) = Y\}$. That is, $\mathsf{Preim}(k, \ell, Y)$ contains the length ℓ preimages of Y under key k. We also define the following function related to any F with $\mathit{Rng} = \{0,1\}^\eta$ for $\eta > 0$:

$$\delta(k, \ell, Y) = \left| \frac{|\mathsf{Preim}(k, \ell, Y)| - 2^{\ell - \eta}}{2^\ell} \right| .$$

The δ function measures how far a particular preimage set deviates from the case in which F_k is regular (that is, $0 \leq \delta(k, \ell, Y) \leq 1 - 2^{-\eta}$). We define $\Delta_k = \max\{\delta(k, \ell, Y)\}$, where the maximum is taken over all choices of ℓ and Y, and we say a function family F is Δ-regular if

$$\sum_{k \in \mathcal{K}} p_k \Delta_k \leq \Delta$$

where $p_k = \Pr[k = k' : k' \xleftarrow{\$} \mathcal{K}]$. Intuitively, this measures the average (over keys) maximum deviation from regularity that F exhibits.

Trapdoor one-way permutations. Let $\mathbb{F}$ be a *trapdoor permutation generator*: on input 1^k it outputs a trapdoor permutation pair (f, f^{-1}) where f: $\{0,1\}^k \to \{0,1\}^k$ and $f^{-1}(f(X)) = X$. We write $\mathsf{Time}_f(k)$ or $\mathsf{Time}_{f^{-1}}(k)$ for the the worst case time to compute f or f^{-1} for any $(f, f^{-1}) \in \mathbb{F}(1^k)$. The one-way advantage of an adversary $\mathcal{A}$ against $\mathbb{F}$ for security parameter k is defined by

$$\mathbf{Adv}^{\mathrm{owf}}_{\mathbb{F},k}(\mathcal{A}) = \Pr\left[f(X) = f(X') \ : \ \begin{array}{c} (f, f^{-1}) \xleftarrow{\$} \mathbb{F}(1^k); X \xleftarrow{\$} \{0,1\}^k; \\ Y \leftarrow f(X); X' \xleftarrow{\$} \mathcal{A}(f, Y) \end{array} \right].$$

The RSA and Rabin function families are conjectured to allow generation of secure trapdoor permutations [32, 28, 29]. We also define a generalization of the owf security, which will be useful later in our proof of the TE construction (Section 7). It captures a game in which an adversary adaptively receives multiple images, learns some of their preimages via an inversion oracle, and then attempts to invert one of the (remaining) images. Formally, we define the some-point one-way advantage of an adversary $\mathcal{A}$ against $\mathbb{F}$ for security parameter k by

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

The image oracle PI, when invoked (on no input), chooses a novel random point $X' \in \mathit{Dom}$, 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 PI and otherwise

it returns $\perp$ (i.e., $\mathcal{B}$ only gets to invert points already returned by PI). The adversary wins if it can find a preimage X such that Y was returned from PI and Y was not queried to Inv. The following lemma establishes that a function is secure in the spowf sense as long as it is secure in the owf sense. Its proof, by a straightforward hybrid argument, is given in Appendix A.

Lemma 2.1 *Let k be a security parameter and $\mathbb{F}$ a trapdoor permutation generator. Let $\mathcal{A}$ be a spowf adversary against $\mathbb{F}$ that runs in time t and makes at most (q_p, q_i) queries to its two oracles (necessarily $q_p \geq q_i$). Then there exists a owf adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathrm{spowf}}_{\mathbb{F},k}(\mathcal{A}) = q_p \cdot \mathbf{Adv}^{\mathrm{owf}}_{\mathbb{F},k}(\mathcal{B})\,.$$

Moreover, $\mathcal{B}$ runs in time $t' \leq t + cq_p\mathsf{Time}_f(1^k) + cq_i \log q_p$ for a small, absolute constant c. $\square$

Code-based games. We utilize code-based games [4] to formalize our indifferentiability security definitions and within our proofs. A game consists of an **Initialize** procedure, zero or more oracle procedures, and zero or more subroutines. All of a game's variables are global, implicitly typed, and initialized to zero for integers, everywhere $\perp$ for associative arrays, the empty string for bit strings, and false for bad flags. A game is executed with an adversary $\mathcal{A}$ as follows. The **Initialize** procedure is executed, followed by running $\mathcal{A}$ on input the value(s) returned by **Initialize**. The adversary, which has its own local variables, can make calls to the game's oracle procedures, passing values from some finite domains associated with the procedures. (Subroutine procedures cannot be called by the adversary.) Eventually $\mathcal{A}$ halts with some output. For a game G we write $\mathcal{A}^{\mathrm{G}} \Rightarrow y$ to represent the event that $\mathcal{A}$ run with game G outputs the value y. The probability of this event occurring is over the coins used by both $\mathcal{A}$ and G. Let

$$\mathbf{Adv}(\mathcal{A}^G, \mathcal{A}^H) = \Pr\left[\,\mathcal{A}^G \Rightarrow 1\,\right] - \Pr\left[\,\mathcal{A}^H \Rightarrow 1\,\right]$$

for any adversary A and games G, H.

Ideal ciphers. For integers $k, n > 0$, a blockcipher $E\colon\ \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ 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 $\mathrm{BC}(k,n)$, 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.

Random functions and random injections. Let $\mathit{Dom}, \mathit{Rng} \subseteq \{0,1\}^*$. Recall that a function $f\colon\ \mathit{Dom} \to \mathit{Rng}$ is injective if $f(X) = f(X')$ implies that $X = X'$. (Necessarily for an injection $|\mathit{Dom}| \leq |\mathit{Rng}|$.) For simplicity, we only consider injections with constant stretch $\tau \geq 0$: if $X \in \mathit{Dom}$ then $|f(X)| = |X| + \tau$. The following subroutines implement a *random function*, a *random function with constant stretch*, and a *random injection with constant stretch*.

subroutine $\mathsf{RF}_{\mathit{Dom},\mathit{Rng}}(i, X)$
If $\mathtt{F}[i,X] = \perp$ then
  $\mathtt{F}[i,X] \xleftarrow{\$} \mathit{Rng}$
Ret $\mathtt{F}[i,X]$

subroutine $\mathsf{RF}^*_{\mathit{Dom},\tau}(i, X)$
If $\mathtt{F}[i,X] = \perp$ then
  $\mathtt{F}[i,X] \xleftarrow{\$} \{0,1\}^{|X|+\tau}$
Ret $\mathtt{F}[i,X]$

subroutine $\mathsf{RI}^*_{\mathit{Dom},\tau}(i, X)$
$\ell \leftarrow |X| + \tau$
If $\mathtt{I}[i,X] = \perp$ then
  $\mathtt{I}[i,X] \xleftarrow{\$} \{0,1\}^\ell \backslash \mathtt{R}[i,\ell]$
  $\mathtt{R}[i,\ell] \xleftarrow{\cup} \mathtt{I}[i,X]$
Ret $\mathtt{I}[i,X]$

The tables $\mathtt{F}$ and $\mathtt{I}$ are initially everywhere set to $\perp$ and the table $\mathtt{R}$ is initially everywhere $\emptyset$. The argument i will be used to distinguish between different instances of random functions (resp. in-

procedure Initialize	**procedure** $\mathcal{O}(M)$	**procedure** $\mathcal{P}_i(X)$	Game CONS_F
Ret $K \xleftarrow{\$} \mathcal{K}$	Ret $F^{f_1,\ldots,f_l}(M)$	Ret $f_i(X)$	

procedure Initialize	**procedure** $\mathcal{O}(M)$	**procedure** $\mathcal{P}_i(X)$	Game $\mathrm{PRO}_{F,\mathcal{S}}$
$K \xleftarrow{\$} \mathcal{K}$; $\mathcal{SI}(K)$	Ret $\mathsf{RF}_{Dom,Rng}(0, M)$	Ret $\mathcal{S}_i(X)$	
Ret K			

procedure Initialize	**procedure** $\mathcal{O}(M)$	**procedure** $\mathcal{P}_i(X)$	Game $\mathrm{PRIO}_{F,\mathcal{S}}$
$K \xleftarrow{\$} \mathcal{SI}$	Ret $\mathsf{RI}^*_{Dom,Rng}(0, M)$	Ret $\mathcal{S}_i(X)$	
Ret K			

Figure 2: Games CONS, PRO, and PRIO used to define indifferentiability for cryptographic functions. Here $\mathcal{P}_i$ is defined for all $i \in [1\,..\,l]$ where l is the number of (idealizable) primitives used by the cryptographic function C.

jections). We will write $f_i = \mathsf{RF}_{Dom,Rng}(i, \cdot)$ or $f_i = \mathsf{RF}^*_{Dom,\tau}(i, \cdot)$ or $\mathcal{I}_i = \mathsf{RI}^*_{Dom,\tau}(i, \cdot)$ to identify a subroutine that works as specified above, sometimes omitting $(i, \cdot)$ when only discussing a single instance. We will write $\mathsf{RF}_{Dom,r}$ if $Rng = \{0,1\}^r$ and $\mathsf{RI}^*_{d,\tau}$ if $Dom = \{0,1\}^d$. A random oracle is a random function that is publically accessible by all parties. Similarly a random injection oracle is a random injection that is publically accessible by all parties.

PROs AND PRIOs. The notion of indifferentiability [23] is a generalization of conventional indistinguishability [18]. It facilitates reasoning about the ability of constructions to emulate some idealized functionality (e.g., a random oracle) in settings where the construction itself utilizes public, idealized components (e.g., an ideal cipher or fixed-input-length (FIL) random oracle). We adapt the formalization of indifferentiability from [13, 2] (to a code-based games setting) to formalize definitions of security for pseudorandom oracles and pseudorandom injective oracles.

A *simulator* $\mathcal{S} = (\mathcal{SI}, \mathcal{S}_1, \ldots, \mathcal{S}_l)$ is a tuple of game subroutines. The (optional) routine $\mathcal{SI}$ will be used for initialization (whenever omitted, it is implicitly instantiated with a no-op). Routines $\mathcal{S}_1, \ldots, \mathcal{S}_l$ implement various oracle functionalities. Note that a simulator $\mathcal{S}$ will be used within a game, and thus maintains state across oracle calls. Moreover, each subroutine's variables (being global) are accessible by the other subroutines. Note however, that we require that $\mathcal{S}$ be defined independently of any particular game it is used within, meaning that it cannot directly access the rest of the enclosing game's variables.

Let $F = (\mathcal{K}, F, Dom, Rng, \tau)$ be a cryptographic function and $\mathcal{S} = (\mathcal{SI}, \mathcal{S}_1, \ldots, \mathcal{S}_l)$ be a simulator. We assume that the number of (black-box) primitives used by F and the number of oracles implemented by $\mathcal{S}$ are equal (to l). Figure 2 shows the games CONS and PRO. These games expose $l + 1$ oracles. We define the pro advantage of an adversary $\mathcal{A}$ against F by

$$\mathbf{Adv}^{\mathrm{pro}}_{F,\mathcal{S}}(\mathcal{A}) = \mathbf{Adv}(\mathcal{A}^{\mathrm{CONS}_F}, \mathcal{A}^{\mathrm{PRO}_{F,\mathcal{S}}}) = \Pr\left[\,\mathcal{A}^{\mathrm{CONS}_F} \Rightarrow 1\,\right] - \Pr\left[\,\mathcal{A}^{\mathrm{PRO}_{F,\mathcal{S}}} \Rightarrow 1\,\right].$$

Similarly we define the prio advantage of an adversary $\mathcal{A}$ against F by

$$\mathbf{Adv}^{\mathrm{prio}}_{F,\mathcal{S}}(\mathcal{A}) = \mathbf{Adv}(\mathcal{A}^{\mathrm{CONS}_F}, \mathcal{A}^{\mathrm{PRIO}_{F,\mathcal{S}}}) = \Pr\left[\,\mathcal{A}^{\mathrm{CONS}_F} \Rightarrow 1\,\right] - \Pr\left[\,\mathcal{A}^{\mathrm{PRIO}_{F,\mathcal{S}}} \Rightarrow 1\,\right]$$

where game PRIO is defined in Figure 2. Note that here the simulator is allowed to choose the key for the cryptographic function. This weakening of the security model (as compared to having the

initialization choose it) is necessary for our proof of the TE construction in Section 7.

3 The MCM Construction

Fix numbers $\eta > 0$ and $\tau \geq 0$ and $L > 0$. Let $H = (\mathcal{K}_H, H, \mathcal{M}_H, \eta)$ be a cryptographic function for $\mathcal{M}_H = \{0,1\}^{\leq L}$ (e.g. L might be 2^{64}). Let $\mathcal{M} = \{0,1\}^{\leq L'}$ for $L' = L - \tau$. Let $\mathcal{E}_1 = (\mathcal{K}_1, \mathcal{E}_1, \mathcal{M}, \mathcal{M}_H, \tau)$ be an injective cryptographic function mapping, with stretch τ, points in $\mathcal{M}$ to points in $\mathcal{M}_H$. Let $\mathcal{E}_2 = (\mathcal{K}_2, \mathcal{E}_2, \eta, \eta+\tau, \tau)$ be an injective cryptographic function mapping, with stretch τ, points in $\{0,1\}^{\eta}$ to points in $\{0,1\}^{\eta+\tau}$. Then $\mathsf{MCM}[\mathcal{E}_1, H, \mathcal{E}_2] = (\mathcal{K}, \mathcal{H}, \mathcal{M}, \eta+\tau)$ is the construction that works as follows. The key generation algorithm $\mathcal{K}$ runs $k_1 \xleftarrow{\$} \mathcal{K}_1$; $k \xleftarrow{\$} \mathcal{K}_H$; $k_2 \xleftarrow{\$} \mathcal{K}_2$ and returns (k_1, k, k_2). Hashing for key $K = (k_1, k, k_2)$ and message $M \in \mathcal{M}$ is computed by

$$\mathcal{H}_K(M) = \mathcal{E}_2(k_2, H(k, \mathcal{E}_1(k_1, M)))\,.$$

Overloading our notation, if $\mathcal{I}_1 = \mathsf{RI}^*_{\mathcal{M},\tau}(1,\cdot)$ and $\mathcal{I}_2 = \mathsf{RI}^*_{\eta,\tau}(2,\cdot)$ then $\mathsf{MCM}[\mathcal{I}_1, H, \mathcal{I}_2] = (\mathcal{K}_H, \mathcal{H}, \mathcal{M}, \eta+\tau)$ is the cryptographic function with key generation algorithm equivalent to that of H and hashing defined by

$$\mathcal{H}(k, M) = \mathcal{I}_2(H(k, \mathcal{I}_1(M)))$$

where computation is done using oracle access to $\mathcal{I}_1$ and $\mathcal{I}_2$. Notice that τ is also the number of bits beyond η needed to hold an MCM hash value. Ideally $\tau = 0$, in which case $\mathcal{E}_2$ would necessarily be a permutation. We have the following theorem, which states that $\mathcal{H}$ inherits the collision-resistance of H.

Theorem 3.1 *Fix $\eta > 0$, $\tau \geq 0$, $L > 0$, and let $\mathcal{M} = \{0,1\}^{\leq L-\tau}$ and $\mathcal{M}_H = \{0,1\}^{\leq L}$. Let $H = (\mathcal{K}_H, H, \mathcal{M}_H, \eta)$ be a cryptographic function and let $\mathcal{E}_1 = (\mathcal{K}_1, \mathcal{E}_1, \mathcal{M}, \mathcal{M}_H, \tau)$ and $\mathcal{E}_2 = (\mathcal{K}_2, \mathcal{E}_2, \eta, \eta+\tau, \tau)$ be injective cryptographic functions. Let $\mathsf{MCM}[\mathcal{E}_1, H, \mathcal{E}_2] = (\mathcal{K}, \mathcal{H}, \mathcal{M}, \eta+\tau)$. 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*

$$\mathbf{Adv}^{\mathrm{cr}}_{\mathcal{H}}(\mathcal{A}) = \mathbf{Adv}^{\mathrm{cr}}_{H}(\mathcal{B})$$

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

Proof: Let $\mathcal{B}$ be the adversary that behaves as follows. On input key k it runs $k_1 \xleftarrow{\$} \mathcal{K}_1$ and $k_2 \xleftarrow{\$} \mathcal{K}_2$, sets $K \leftarrow (k_1, k, k_2)$, and runs $\mathcal{A}(K)$. Adversary $\mathcal{A}$ outputs two messages (X, X'). Then $\mathcal{B}$ outputs $(\mathcal{E}_1(k_1, X), \mathcal{E}_1(k_1, X'))$. We have that if $\mathcal{H}(K, X) = \mathcal{H}(K, X')$ then because $\mathcal{E}_1$ and $\mathcal{E}_2$ are injections, necessarily $H(k, \mathcal{E}_1(k_1, X)) = H(k, \mathcal{E}_1(k_1, X'))$. Adversary $\mathcal{B}$ runs in time $t' \leq t + 2(c\mu + \mathsf{Time}_{\mathcal{E}_1}(\mu))$ where c is an absolute constant. ▌

We point out that similar theorems can be given for several other hash function properties, including target collision-resistance (TCR, or eSec), preimage resistance, and always preimage resistance (aPre) [34][2]. The next theorem captures that MCM is a PRO if both $\mathcal{E}_1$ and $\mathcal{E}_2$ are modeled as random injections.

Theorem 3.2 *Fix $\eta > 0$, $\tau \geq 0$, $L > 0$, and let $\mathcal{M} = \{0,1\}^{\leq L-\tau}$ and $\mathcal{M}_H = \{0,1\}^{\leq L}$. Let $H = (\mathcal{K}_H, H, \mathcal{M}_H, \eta)$ be a Δ-regular cryptographic function, $\mathcal{I}_1 = \mathsf{RI}^*_{\mathcal{M},\tau}(1,\cdot)$, and $\mathcal{I}_2 = \mathsf{RI}^*_{\eta,\tau}(2,\cdot)$.*

[2] Although it is unclear how one would prove that MCM preserves the other notions from [34], specifically everywhere preimage resistance (ePre), second-preimage resistance (Sec) and always second-preimage resistance (aSec).

Let $\mathsf{MCM}[\mathcal{I}_1, H, \mathcal{I}_2] = (\mathcal{K}_H, \mathcal{H}, \mathcal{M}, \eta + \tau)$. *Let* ν *be the minimal message length in* $\mathcal{M}$. *Let* $\mathcal{A}$ *be an adversary that runs in time* t *and makes at most* (q_0, q_1, q_2) *queries to its three oracles with the maximal query length being* μ *bits. Let* $q = q_0 + q_1 + q_2$. *Then there exists an adversary* $\mathcal{B}$ *such that*

$$\mathbf{Adv}^{\mathrm{pro}}_{\mathcal{H},\mathcal{S}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{cr}}_{H}(\mathcal{B}) + q^2\left(\frac{1}{2^{\eta}} + \frac{1}{2^{\nu+\tau}} + \Delta\right) \tag{1}$$

where the simulator $\mathcal{S}$ *is specified in Figure 3. Let* c *be an absolute constant. Then,* $\mathcal{S}$ *runs in time* $t_{\mathcal{S}} \leq c\mu(q_1\mathsf{Time}_H(\mu) + (q_1+q_2)\log q_1)$ *and makes at most* $\min\{q_1, q_2\}$ *oracle queries. Adversary* $\mathcal{B}$ *runs in time at most* $t_{\mathcal{B}} \leq t + c'\mu(q\mathsf{Time}_H(\mu) + q\log q)$. $\square$

As long as $\mathcal{E}_1$ and $\mathcal{E}_2$ are PRIOs we can securely replace them by actual random injections (as per the composition theorem of [23], though see Section 8). Then, Theorem 3.2 states that no adversary can differentiate between a random oracle and the construction unless it is given sufficient time to break the collision-resistance of H, allowed to make approximately $2^{(\min\{\eta,\nu+\tau\})/2}$ queries, or H is not sufficiently close to regular for all but a negligible fraction of the keys. 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. The proof of Theorem 3.2 is given in Section 5.

4 Insecurity of Other Approaches

Here we give just a brief investigation of several alternative approaches to 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: $\{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ 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: $\{0,1\}^* \to (\{0,1\}^n)^+$ 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. [11] 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.[3]

[3] Hopwood and Wagner noted (in postings on sci.crypt) that one could exhibit good PRPs that would make finding

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 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 collision.

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 [13], but we state a version of it here for completeness. Let $\mathcal{H} = \mathsf{CM}[H, \mathcal{I}_2]$ be this modified construction for $\mathcal{I}_2 = \mathsf{RI}^*_{\eta,\eta+\tau}$, i.e. $\mathcal{H}(M) = \mathcal{I}_2(H(M))$. Now we show that $\mathcal{H}$ is easily differentiable from a true random oracle $\mathcal{R} = \mathsf{RF}_{\mathcal{M}_H,\eta+\tau}$. Let $\mathcal{A}$ be an adversary that queries it's first oracle on a uniformly selected message $M \in \mathcal{M}_H$ of some length ℓ. Let the returned value be C. Now the adversary queries its second oracle (being either $\mathcal{I}_2$ or a simulator) on $H_K(M)$. 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 $\Pr\left[\mathcal{A}^{\mathcal{H},\mathcal{I}_2} \Rightarrow 1\right] = 1$. On the other hand, $\Pr\left[\mathcal{A}^{\mathcal{R},\mathcal{S}} \Rightarrow 1\right]$ is bounded by the advantage of another adversary (related to the simulator) in breaking the one-wayness of H; essentially, the simulator is unable to query $\mathcal{R}$ on M without inverting $H_K(M)$.

Allowing $\mathcal{E}_1, \mathcal{E}_2$ to be invertible. Our formalization of PRIOs ensure that constructions meeting the goal are *not* invertible. Thus, objects that are invertible do not meet the goal. It remains an open question whether MCM is, in fact, secure under easy-to-invert mixing steps.

5 Proof of Theorem 3.2

The simulator. Let $\mathcal{R} = \mathsf{RF}_{\mathcal{M},\eta+\tau}(0,\cdot)$ be a random oracle. Figure 3 depicts a simulator $\mathcal{S} = (\mathcal{SI}, \mathcal{S}_1, \mathcal{S}_2)$. The initialization subroutine $\mathcal{SI}$ just records the key for H. The subroutine $\mathcal{S}_1$ implements a random injection for bit strings in $\mathcal{M}$. The subroutine $\mathcal{S}_2$, when queried on bit string $Y \in \{0,1\}^\eta$, checks if $\mathcal{S}_1$ already maps a string M to a preimage of Y under H_k. If such an M exists, then the simulator queries $\mathcal{R}$ on M and the output of $\mathcal{S}_2(Y)$ is programmed to match the value returned by $\mathcal{R}$. Otherwise, a random string of length $\eta + \tau$ is returned.

We have that $\mathcal{S}$ makes at most $\min\{q_1, q_2\}$ queries to its oracle when run in conjunction with any pro adversary making at most (q_0, q_1, q_2) queries to its oracles. (We take the minimum because the simulator will query its oracle only when two queries by the adversary to its second and third oracles are associated appropriately.) The simulator's $\mathcal{S}_1$ subroutine requires time proportional to $\mathsf{Time}_H(\mu) + \log q_1$. (This is the time needed to compute H_k, to implement the random injection, and the time to build the `YtoM` table.) The simulator's $\mathcal{S}_2$ subroutine requires time proportional to $\log q_1$ (this is the time needed to do lookups in `YtoM`). Thus, $t_{\mathcal{S}} \leq c\mu(\mathsf{Time}_H(\mu) \cdot q_1 + (q_1 + q_2)\log q_1)$ for c a small, absolute constant.

The proof. The proof is captured by two main lemmas. The first lets us simplify the simulator by showing that we can focus on a certain class of adversaries without loss of generality. The second

collisions in the twenty functions [5] trivial.

subroutine $\mathcal{SI}(k')$:	**subroutine** $\mathcal{S}_1(M)$	**subroutine** $\mathcal{S}_2(Y)$
$k \leftarrow k'$	$X \leftarrow \mathsf{RI}^*_{\mathcal{M},\tau}(1, M)$	If $\mathtt{YtoM}[Y] \neq \perp$ then
	$\mathtt{YtoM}[H_k(X)] \leftarrow M$	Ret $\mathcal{R}(\mathtt{YtoM}[Y])$
	Ret X	Ret $Z \xleftarrow{\$} \{0,1\}^{\eta+\tau}$

subroutine $\mathcal{SSI}(k')$	**subroutine** $\mathcal{SS}_1(M)$	**subroutine** $\mathcal{SS}_2(Y)$
$k \leftarrow k'$	Ret $X \leftarrow \mathsf{RI}^*_{\mathcal{M},\tau}(1, M)$	Ret $Z \xleftarrow{\$} \{0,1\}^{\eta+\tau}$

Figure 3: **(Top)** Simulator $\mathcal{S} = (\mathcal{SI}, \mathcal{S}_1, \mathcal{S}_2)$ used in the proof of Theorem 3.2. The simulator expects access to a subroutine $\mathcal{R} = \mathsf{RF}_{\mathcal{M},\eta+\tau}$, which implements a random oracle. **(Bottom)** The simplified simulator $\mathcal{SS} = (\mathcal{SSI}, \mathcal{SS}_1, \mathcal{SS}_2)$.

lemma bounds the probability of success of any pro-adversary in this class against the simplified simulator.

Let the simplified simulator $\mathcal{SS} = (\mathcal{SSI}, \mathcal{SS}_1, \mathcal{SS}_2)$ be defined as shown in Figure 3. Here $\mathcal{SS}_1$ implements a random injection and $\mathcal{SS}_2$ always returns a string of random bits. (Recall that pointless queries are disallowed, so $\mathcal{SS}_2$ need not perform consistency checks.) Note that the key received by the simulator in $\mathcal{SSI}$ is never actually used, so we could technically dispense with $\mathcal{SSI}$ completely.

Let $\mathcal{A}$ be any pro adversary against $\mathcal{H}$ that makes at most q queries. Consider the query transcript $(ty_1, Q_1, R_1)_{\mathcal{A}}, \ldots, (ty_t, Q_t, R_t)_{\mathcal{A}}$ that results from running $\mathcal{A}$ within game $\mathrm{CONS}_{\mathcal{H}}$ or $\mathrm{PRO}_{\mathcal{H},\mathcal{S}}$. For $1 \le i \le t$, $ty_i \in \{0,1,2\}$ specifies to which oracle (numbered left to right) the ith query was made, the query being $Q_i \in \mathcal{M} \cup \{0,1\}^\nu$, with response $R_i \in \mathcal{M}_H \cup \{0,1\}^{\nu+\tau}$. We show that it is sufficient to consider an adversary $\mathcal{A}$ which is restricted in the types of oracle queries it makes. Let a *construction-respecting* adversary be such that the transcript resulting in its interaction with any triple of oracles does not have entries $(1, Q_i, R_i)$ and $(2, Q_j, R_j)$ with $i < j$ and $Q_j = H_k(R_i)$. In particular, this means that such an adversary never queries the second oracle on a value, gets the response, hashes it with H_k, and then queries the third oracle on the resulting value. Such a pair of queries will be called *construction disrespecting.*

Lemma 5.1 *Let $\mathcal{A}$ be a pro adversary against $\mathcal{H}$ running in time at most t and making at most q queries to all of its oracles with the combined length of all queries at most μ. Then there exists a construction-respecting adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathrm{pro}}_{\mathcal{H},\mathcal{S}}(\mathcal{A}) = \mathbf{Adv}^{\mathrm{pro}}_{\mathcal{H},\mathcal{SS}}(\mathcal{B})$$

where $\mathcal{B}$ runs in time at most $t_{\mathcal{B}} \le t + c\mu(q \cdot \mathsf{Time}_H(\mu) + q \log q)$ and makes at most q queries.

Proof: We construct an adversary $\mathcal{B}$ that runs $\mathcal{A}$ and incorporates the checks done by simulator $\mathcal{S}$ for various oracle queries. Namely let $\mathcal{B}$, given oracles $\mathcal{O}'_0, \mathcal{P}'_1, \mathcal{P}'_2$ and run on input k, work as follows. It runs $\mathcal{A}(k)$ and answer $\mathcal{A}$'s oracle queries as follows.

query $\mathcal{O}_0(M)$:	**query** $\mathcal{P}_1(M)$:	**query** $\mathcal{P}_2(Y)$:
Ret $\mathcal{O}'_0(M)$	$X \leftarrow \mathcal{P}'_1(M)$	If $\mathtt{YtoM}[Y] \neq \perp$ then
	$\mathtt{YtoM}[H_k(X)] \leftarrow M$	Ret $\mathcal{O}'_0(\mathtt{YtoM}[Y])$
	Ret X	Ret $\mathcal{P}'_2(Y)$

The table $\mathtt{YtoM}$ is initially everywhere bottom. When $\mathcal{A}$ halts with output bit b, adversary $\mathcal{B}$

outputs b. Adversary $\mathcal{B}$ runs in time $t + c\mu(q\mathsf{Time}_H(\mu) + q\log q)$ where c is an absolute, small constant and $\log q$ accounts for the time required for `YtoM`. We now justify that

$$\Pr\left[\mathcal{A}^{\mathrm{CONS}_{\mathcal{H}}} \Rightarrow 1\right] = \Pr\left[\mathcal{B}^{\mathrm{CONS}_{\mathcal{H}}} \Rightarrow 1\right], \text{ and} \quad (2)$$

$$\Pr\left[\mathcal{A}^{\mathrm{PRO}_{\mathcal{H},\mathcal{S}}} \Rightarrow 1\right] = \Pr\left[\mathcal{B}^{\mathrm{PRO}_{\mathcal{H},\mathcal{S}\mathcal{S}}} \Rightarrow 1\right]. \quad (3)$$

For (2) we have by construction that both $\mathcal{A}$ and $\mathcal{B}$ behave identically until $\mathcal{A}$ induces a construction-disrespecting pair of queries. Consider the first such pair made by $\mathcal{A}$, letting the resulting transcript entries be $(1, Q_i, R_i)_{\mathcal{A}}$ and $(2, H_k(R_i), R_j)_{\mathcal{A}}$ where $i < j$. If $\mathcal{A}$ is interacting with the oracles $(\mathcal{H}, \mathcal{I}_1, \mathcal{I}_2)$ in game $\mathrm{CONS}_{\mathcal{H}}$, then it receives responses $R_i = \mathcal{I}_1(Q_i)$ and $R_j = \mathcal{I}_2(H_k(R_i)) = \mathcal{I}_2(H_k(\mathcal{I}_1(Q_i)))$. If, on the other hand, $\mathcal{A}$ is running within $\mathcal{B}$ (which in turn has the same oracles $(\mathcal{H}, \mathcal{I}_1, \mathcal{I}_2)$), then instead of the query $(2, H_k(R_i), R_j)_{\mathcal{A}}$, the transcript will have an entry $(0, Q_j, R'_j)_{\mathcal{B}}$ in its place. Here, $Q_j = Q_i$ and, by the construction of MCM, $R'_j = \mathcal{I}_2(H_k(\mathcal{I}_1(Q_i)))$. Therefore in both cases $\mathcal{A}$ receives identically distributed oracle responses. Repeating this reasoning for each pair of construction-disrespecting queries justifies (2).

For (3) we first show that the probability that $\mathcal{A}^{\mathrm{PRO}_{\mathcal{H},\mathcal{S}}} \Rightarrow 1$ is equal to the probability that $\mathcal{B}^{\mathrm{PRO}_{\mathcal{H},\mathcal{S}}} \Rightarrow 1$. The argument is similar to the one just given. In short, consider a pair of construction-disrespecting queries and their transcript entries $(1, Q_i, R_i)_{\mathcal{A}}$ and $(2, H_k(R_i), R_j)_{\mathcal{A}}$ for $i < j$. If $\mathcal{A}$ is interacting with $(\mathcal{R}, \mathcal{S}_1, \mathcal{S}_2)$ it receives responses R_i computed by $\mathsf{RI}^*_{\mathcal{M},\tau}(1, Q_i)$ and $R_j = \mathcal{R}(Q_i)$ (because the conditional in $\mathcal{S}_2$ will evaluate to true for such a query). If $\mathcal{A}$ is running within $\mathcal{B}$, then $\mathcal{B}$ replaces $(2, H_k(R_i), R_j)_{\mathcal{A}}$ with $(0, Q_j, R_j)_{\mathcal{B}}$ where $Q_j = Q_i$ and $R_j = \mathcal{R}(Q_i)$. Thus the events occur with the same probability. Now we note that, for any construction-respecting adversary $\mathcal{B}$, execution of $\mathcal{B}^{\mathrm{PRO}_{\mathcal{H},\mathcal{S}}}$ never causes the conditional statement in $\mathcal{S}_2$ to evaluate to true. Thus we can replace $\mathcal{S}$ with $\mathcal{S}\mathcal{S}$, justifying (3). ▌

Lemma 5.2 *Let $\mathcal{A}$ be a construction-respecting pro adversary against $\mathcal{H}$ running in time at most t and making at most (q_0, q_1, q_2) queries to its oracles with the maximum query length being μ bits. Then there exists a cr adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathrm{pro}}_{\mathcal{H},\mathcal{S}\mathcal{S}}(\mathcal{A}) \le \mathbf{Adv}^{\mathrm{cr}}_{H}(\mathcal{B}) + \frac{(q_0+q_2)^2}{2^{\eta+\tau}} + \frac{(q_0+q_1)^2}{2^{\nu+\tau}} + (q_0+q_1)q_2\left(\frac{1}{2^{\eta}} + \Delta\right) \quad (4)$$

where $\mathcal{B}$ runs in time at most $t_{\mathcal{B}} \le t + c\mu(q_0 \cdot \mathsf{Time}_H(\mu) + (q_0+q_1)\log(q_0+q_1) + q_0\log q_0 + q_2)$. ▌

Proof: First we lift the injectivity constraint of the third oracle, i.e. $\mathcal{I}_2$, in game $\mathrm{CONS}_{\mathcal{H}}$, incurring a standard birthday-bound loss. That is let $\mathrm{CONS}'_{\mathcal{H}}$ be the game that is the same as $\mathrm{CONS}_{\mathcal{H}}$ except that oracle $\mathcal{P}_2$ now implements a random function $\mathsf{RF}_{\eta,\eta+\tau}(2,\cdot)$. Then,

$$\Pr\left[\mathcal{A}^{\mathrm{CONS}_{\mathcal{H}}} \Rightarrow 1\right] \le \Pr\left[\mathcal{A}^{\mathrm{CONS}'_{\mathcal{H}}} \Rightarrow 1\right] + \frac{(q_1+q_3)^2}{2^{\eta+\tau}} \quad (5)$$

where here the numerator of the last term is the maximum number of times a range point of $\mathcal{I}_2$ is sampled. Game G0 (Figure 4, boxed statements included) implements $\mathrm{CONS}'_{\mathcal{H}}$. The table `F` is used to build the random function $\mathsf{RF}_{\eta,\eta+\tau}(2,\cdot)$. (The consistency checks on lines 014 and 033 ensure that game G0 implements a random function for the third oracle.) The random injection $\mathcal{I}_1$ is implemented directly using a subroutine $\mathsf{RI}^*_{\mathcal{M},\tau}(1,\cdot)$ called on lines 011 and 021. This justifies that

$$\Pr\left[\mathcal{A}^{\mathrm{CONS}'_{\mathcal{H}}} \Rightarrow 1\right] = \Pr\left[\mathcal{A}^{\mathrm{G0}} \Rightarrow 1\right]. \quad (6)$$

procedure Initialize
000 Ret $k \xleftarrow{\$} \mathcal{K}_H$

procedure $\mathcal{O}_0(M)$
010 $j \leftarrow j+1$; $M^j \leftarrow M$
011 $X^j \leftarrow \mathsf{RI}^*_{\mathcal{M},\tau}(1, M^j)$
012 $Y^j \leftarrow H_k(X^j)$; $C^j \xleftarrow{\$} \{0,1\}^{\eta+\tau}$
013 If $\mathtt{F}[Y^j] \neq \perp$ then
014 bad $\leftarrow$ true; $\boxed{C^j \leftarrow \mathtt{F}[Y^j]}$
015 Ret $\mathtt{F}[Y^j] \leftarrow C^j$

procedure $\mathcal{P}_1(M)$ $\boxed{\text{G0}}$ G1
020 $j \leftarrow j+1$; $M^j \leftarrow M$
021 $X^j \leftarrow \mathsf{RI}^*_{\mathcal{M},\tau}(1, M^j)$
022 Ret X^j

procedure $\mathcal{P}_2(Y)$
030 $j \leftarrow j+1$; $Y^j \leftarrow Y$
031 $C^j \xleftarrow{\$} \{0,1\}^{\eta+\tau}$
032 If $\mathtt{F}[Y^j] \neq \perp$ then
033 bad $\leftarrow$ true; $\boxed{C^j \leftarrow \mathtt{F}[Y^j]}$
034 Ret $\mathtt{F}[Y^j] \leftarrow C^j$

procedure Initialize
200 Ret $k \xleftarrow{\$} \mathcal{K}_H$

procedure $\mathcal{O}_0(M)$
210 $j \leftarrow j+1$; $M^j \leftarrow M$
211 $X^j \leftarrow \mathsf{RI}^*_{\mathcal{M},\tau}(1, M^j)$
212 $Y^j \leftarrow H_k(X^j)$
213 If $\mathtt{F}[Y^j] \neq \perp$ then bad $\leftarrow$ true
214 $\mathtt{F}[Y^j] \leftarrow 1$
215 Ret $C^j \xleftarrow{\$} \{0,1\}^{\eta+\tau}$

procedure $\mathcal{P}_1(M)$ G2
220 $j \leftarrow j+1$; $M^j \leftarrow M$
221 $X^j \leftarrow \mathsf{RI}^*_{\mathcal{M},\tau}(1, M^j)$
222 Ret X^j

procedure $\mathcal{P}_2(Y)$
230 $j \leftarrow j+1$; $Y^j \leftarrow Y$
231 If $\mathtt{F}[Y^j] \neq \perp$ then bad $\leftarrow$ true
232 $\mathtt{F}[Y^j] \leftarrow 1$
233 Ret $C^j \xleftarrow{\$} \{0,1\}^{\eta+\tau}$

procedure Initialize
300 Ret $k \xleftarrow{\$} \mathcal{K}_H$

procedure $\mathcal{O}_0(M)$
310 $j \leftarrow j+1$; $M^j \leftarrow M$
311 $X^j \leftarrow \boxed{\mathsf{RI}^*_{\mathcal{M},\tau}(1, M^j)}$ $[\![\mathsf{RF}^*_{\mathcal{M},\tau}(1, M^j)]\!]$
312 $Y^j \leftarrow H_k(X^j)$
313 If $\mathtt{F}[0,0,Y^j] \neq \perp$ then bad1 $\leftarrow$ true
314 If $\mathtt{F}[0,2,Y^j] \neq \perp$ then bad2 $\leftarrow$ true
315 $\mathtt{F}[0,0,Y^j] \leftarrow \mathtt{F}[0,2,Y^j] \leftarrow 1$
317 Ret $C^j \xleftarrow{\$} \{0,1\}^{\eta+\tau}$

procedure $\mathcal{P}_1(M)$ $\boxed{\text{G3}}$ $[\![\text{G4}]\!]$
320 $j \leftarrow j+1$; $M^j \leftarrow M$
321 $X^j \leftarrow \boxed{\mathsf{RI}^*_{\mathcal{M},\tau}(1, M^j)}$ $[\![\mathsf{RF}^*_{\mathcal{M},\tau}(1, M^j)]\!]$
322 $Y^j \leftarrow H_k(X^j)$
323 If $\mathtt{F}[1,2,Y^j] \neq \perp$ then bad2 $\leftarrow$ true
324 Ret X^j

procedure $\mathcal{P}_2(Y)$
330 $j \leftarrow j+1$; $Y^j \leftarrow Y$
331 If $\mathtt{F}[0,2,Y^j] \neq \perp$ then bad2 $\leftarrow$ true
332 $\mathtt{F}[0,2,Y^j] \leftarrow \mathtt{F}[1,2,Y^j] \leftarrow 1$
333 Ret $C^j \xleftarrow{\$} \{0,1\}^{\eta+\tau}$

Figure 4: Games used in the proof of Theorem 3.2. Initially $j = 0$ and the table $\mathtt{F}$ is everywhere $\perp$.

Game G1 (Figure 4, boxed statements omitted) is exactly like G0 but with the boxed statements removed. In particular this means that $\mathcal{O}_0$ and $\mathcal{P}_2$ always return random bits. Notice that queries to $\mathcal{O}_0$ can determine points under the injection on line 011 (something not possible via a query to the oracle $\mathcal{R}$); however, the adversary learns nothing about these points and therefore they appear fresh if later returned by an $\mathcal{P}_1$ query. Similarly, $\mathcal{O}_0$ queries can set values in the table $\mathtt{F}$ on line 015, but these values do not impact any later random variables in the game (because of the boxed statements are omitted in game G1). Thus, G1 implements the oracles $(\mathcal{R}, \mathcal{SS}_1, \mathcal{SS}_2)$, i.e. game

$\mathrm{PRO}_{\mathcal{H},\mathcal{SS}}$, and so

$$\Pr\left[\mathcal{A}^{\mathrm{PRO}_{\mathcal{H},\mathcal{SS}}} \Rightarrow 1\right] = \Pr\left[\mathcal{A}^{\mathrm{G1}} \Rightarrow 1\right]. \tag{7}$$

The games are also identical-until-bad and so the fundamental lemma of game-playing [4], together with (5), (6), and (7), justifies

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{pro}}_{\mathcal{H},\mathcal{SS}}(\mathcal{A}) &= \Pr\left[\mathcal{A}^{\mathrm{CONS}_{\mathcal{H}}} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathrm{PRO}_{\mathcal{H},\mathcal{SS}}} \Rightarrow 1\right] \\
&\leq \Pr\left[\mathcal{A}^{\mathrm{CONS}'_{\mathcal{H}}} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathrm{PRO}_{\mathcal{H},\mathcal{SS}}} \Rightarrow 1\right] + \frac{(q_1+q_3)^2}{2^{\eta+\tau}} \\
&= \Pr\left[\mathcal{A}^{\mathrm{G0}} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathrm{G1}} \Rightarrow 1\right] + \frac{(q_1+q_3)^2}{2^{\eta+\tau}} \\
&\leq \Pr\left[\mathcal{A}^{\mathrm{G1}} \text{ sets } \mathsf{bad}\right] + \frac{(q_1+q_3)^2}{2^{\eta+\tau}} \qquad (8)
\end{aligned}$$

Game G2, shown in Figure 4, modifies game G1 in the following ways. The random sampling of C^j in $\mathcal{O}_0$ and $\mathcal{P}_2$ are deferred until the end of the procedures. Instead of filling the table F with values, positions are just marked with a 1 to represent having been defined. It is clear that

$$\Pr\left[\mathcal{A}^{\mathrm{G1}} \text{ sets } \mathsf{bad}\right] = \Pr\left[\mathcal{A}^{\mathrm{G2}} \text{ sets } \mathsf{bad}\right]. \tag{9}$$

Game G3 (Figure 4, boxed statements included and barred statements omitted) differs from G2 in that we replace bad with two distinct flags bad1 and bad2. The former is set when two Y^j values defined in $\mathcal{O}_0$ collide (line 313). The latter is set if a Y^j value in $\mathcal{O}_0$ and a Y^j value in $\mathcal{P}_2$ collide (lines 314 and 331) or if a Y^j value chosen in $\mathcal{P}_1$ collides with a Y^j previously chosen in $\mathcal{P}_2$ (line 323). We have that if executing $\mathcal{A}^{\mathrm{G2}}$ on a sequence of random coins leads to bad being set, then executing $\mathcal{A}^{\mathrm{G3}}$ on the same random coins will necessarily lead to either bad1 or bad2 being set. (Note that more sequences of random coins could result in $\mathcal{A}^{\mathrm{G3}}$ setting one of the two flags, because of the line 323.) Thus

$$\begin{aligned}
\Pr\left[\mathcal{A}^{\mathrm{G2}} \text{ sets } \mathsf{bad}\right] &\leq \Pr\left[\mathcal{A}^{\mathrm{G3}} \text{ sets } \mathsf{bad1} \vee \mathcal{A}^{\mathrm{G3}} \text{ sets } \mathsf{bad2}\right] \qquad (10) \\
&\leq \Pr\left[\mathcal{A}^{\mathrm{G3}} \text{ sets } \mathsf{bad1}\right] + \Pr\left[\mathcal{A}^{\mathrm{G3}} \text{ sets } \mathsf{bad2}\right]. \qquad (11)
\end{aligned}$$

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 G3, meaning

$$\Pr\left[\mathcal{A}^{\mathrm{G3}} \text{ sets } \mathsf{bad1}\right] = \mathbf{Adv}^{\mathrm{cr}}_{H}(\mathcal{B}). \tag{12}$$

Let $\mathcal{B}$ be the adversary that, on input k, runs $\mathcal{A}(k)$ and answers $\mathcal{A}$'s oracle queries as in G3. 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). To justify (12), first note that the distributions of random variables in the process $k \xleftarrow{\$} \mathcal{K}_H$; $\mathcal{B}(k)$ and in $\mathcal{A}^{\mathrm{G3}}$ are the same. Suppose $\mathcal{B}$ forces bad1 to be set. Then two values Y^i and Y^j collide for some $i \neq j$. The fact that $\mathcal{A}$ never makes pointless queries implies that the corresponding queries M^i and M^j made to $\mathcal{O}_0$ by $\mathcal{A}$ are not equal. In turn the injectivity of $\mathsf{RI}^*_{\mathcal{M},\tau}(1,\cdot)$ gives that $X^i \neq X^j$.

To bound the probability of bad2 being set, we first perform one more game transition. Game G4 (Figure 4, boxed statements replaced by barred statements) behaves exactly like G3 except we

remove the injectivity of the first mixing step. We relate the games G3 and G4 by the inequality

$$\Pr\left[\,\mathcal{A}^{\mathrm{G3}} \text{ sets } \mathsf{bad2}\,\right] \le \Pr\left[\,\mathcal{A}^{\mathrm{G4}} \text{ sets } \mathsf{bad2}\,\right] + \frac{(q_1+q_2)^2}{2^{\nu+\tau}} \tag{13}$$

which can be justified by a simple birthday argument where q_1+q_2 is the maximum number of range points sampled by $\mathsf{RI}^*_{\mathcal{M},\tau}(1,\cdot)$ in G3.

Now we bound the probability that $\mathsf{bad2}$ is set in G4. We first argue that the probability of $\mathsf{bad2}$ being set is unrelated to the adaptivity of $\mathcal{A}$ and the choice of messages queried by $\mathcal{A}$ to $\mathcal{P}_2$. If $\mathsf{bad2}$ is set on line 314 or 323, it is because of a fresh random choice of Y about which the adversary knows nothing. If $\mathsf{bad2}$ is set on line 331, then the adversarially-chosen value Y collides with a value Y previously chosen. However, because $\mathcal{A}$ is construction-respecting it has no information about Y at the time of the $\mathcal{P}_2$ query. (The only way to learn such a point would be to query $\mathcal{P}_1$ before the $\mathsf{bad2}$-setting query to $\mathcal{P}_2$.). Thus even in this case it's a fresh random value.

We can therefore bound the probability of $\mathsf{bad2}$ being set in G4 by the probability of success in the following combinatorial game. Pick a key $k \xleftarrow{\$} \mathcal{K}_H$. Fix any q_3 points $\mathrm{Y}_1,\ldots,\mathrm{Y}_{q_3}$ from $\{0,1\}^\nu$. Let $\mathcal{Y} = \{\mathrm{Y}_1,\ldots,\mathrm{Y}_{q_3}\}$. Fix any q_1+q_2 numbers $\mathtt{l}_1,\ldots,\mathtt{l}_{q_1+q_2}$ such that $\{0,1\}^{\mathtt{l}_i} \in \mathcal{M}_H$ for $1 \le i \le q_1+q_2$. Finally, choose values $X_i \xleftarrow{\$} \{0,1\}^{\mathtt{l}_i}$ for $1 \le i \le q_1+q_2$. The combinatorial game is "successful" if there exists i,j such that $H_k(X_i) = \mathrm{Y}_j$. In terms of our proof, $\mathcal{A}$ gets to pick the (optimal) Y values and $\mathtt{l}$ values and these will be its $\mathcal{P}_2$ queries and the lengths of it's $\mathcal{O}_0$ and $\mathcal{P}_1$ queries, respectively. We have that

$$\begin{aligned}
\Pr\left[\,\mathcal{A}^{\mathrm{G4}} \text{ sets } \mathsf{bad2}\,\right] &\le \Pr\left[\,H_k(X^i) = Y^j \text{ for some i,j}\,\right] \\
&\le \sum_{i=1}^{q_0+q_1} \sum_{\mathrm{K}\in\mathcal{K}_H} \Pr\left[\,H_{\mathrm{K}}(X_i) \in \mathcal{Y} \,|\, K = \mathrm{K}\,\right] \cdot \Pr\left[\,K = \mathrm{K}\,\right] \\
&= \sum_{i=1}^{q_0+q_1} \sum_{\mathrm{K}\in\mathcal{K}_H} \Pr\left[\,\bigvee_{j=1}^{q_3} X_i \in \mathsf{PreIm}(\mathrm{K},\mathtt{l}_i,\mathrm{Y}_j)\,\right] \cdot p_{\mathrm{K}} \\
&\le \sum_{i=1}^{q_0+q_1} \sum_{\mathrm{K}\in\mathcal{K}_H} \sum_{j=1}^{q_3} \Pr\left[\,X_i \in \mathsf{PreIm}(\mathrm{K},\mathtt{l}_i,\mathrm{Y}_j)\,\right] \cdot p_{\mathrm{K}} \\
&= \sum_{i=1}^{q_0+q_1} \sum_{j=1}^{q_2} \sum_{\mathrm{K}\in\mathcal{K}_H} \frac{|\mathsf{PreIm}(\mathrm{K},\mathtt{l}_i,\mathrm{Y}_j)|}{2^{\mathtt{l}_i}} \cdot p_{\mathrm{K}} \\
&\le \sum_{i=1}^{q_0+q_1} \sum_{j=1}^{q_2} \sum_{\mathrm{K}\in\mathcal{K}_H} \left(\frac{2^{\mathtt{l}_i-\eta}}{2^{\mathtt{l}_i}} + \delta(\mathrm{K},\mathtt{l}_i,\mathrm{Y}_j)\right) \cdot p_{\mathrm{K}} \\
&= \sum_{i=1}^{q_0+q_1} \sum_{j=1}^{q_2} \left[\frac{2^{\mathtt{l}_i-\eta}}{2^{\mathtt{l}_i}} + \sum_{\mathrm{K}\in\mathcal{K}_H} p_{\mathrm{K}} \cdot \delta(\mathrm{K},\mathtt{l}_i,\mathrm{Y}_j)\right] \\
&\le \frac{(q_0+q_1)q_3}{2^\eta} + \sum_{i=1}^{q_0+q_1} \sum_{j=1}^{q_2} \sum_{\mathrm{K}\in\mathcal{K}_H} p_{\mathrm{K}} \cdot \Delta_{\mathrm{K}} \\
&\le \frac{(q_0+q_1)q_3}{2^\eta} + (q_0+q_1)q_2\Delta
\end{aligned}$$

where $p_{\kappa} = \Pr[k = \kappa : k \xleftarrow{\$} \mathcal{K}_H]$. Combining the inequality above with (8), (9), (10), (11), (12), and (13) implies equation (4), the advantage relation given in Lemma 5.2.

To finish the proof of the lemma, we analyze the resources used by the adversary $\mathcal{B}$. Adversary $\mathcal{B}$ runs $\mathcal{A}^{\mathrm{G3}}$. Note, however, that lines 314, 315, 323, 331, and 332 of G3 can be omitted since these only deal with setting bad2. Lines 311 and 321 (implementing the random injection) require at most $(q_0 + q_1)\log(q_0 + q_1)$ time since $q_0 + q_1$ points are sampled. Line 312 requires at most $q_0 \cdot \mathsf{Time}_H(\mu)$ time over the course of the game. Line 313 will require at most $q_0 \log q_0$ time. Thus, $t_{\mathcal{B}} \leq t + c\mu(q_0 \cdot \mathsf{Time}_H(\mu) + (q_0 + q_1)\log(q_0 + q_1) + q_0 \log q_0 + q_2)$ for a small, absolute constant c. ∎

6 Secure Mixing Steps: the TE Construction

We now turn to showing the feasibility of instantiating the mixing steps $\mathcal{E}_1$ and $\mathcal{E}_2$ starting from blockciphers. We note that our eventual construction also works starting from a suitable fixed-input-length random oracle. This would have slight theoretical benefits because it is unknown whether the ROM and ICM are equivalent [17]. However, one might want to utilize blockciphers and the proofs are only rendered more complex when considering invertible components, thus we stick to the former.

We specify a construction that is a PRIO, i.e. indifferentiable from a random injection. Under the composability guarantees of the indifferentiability framework [23] (though see Section 8), the security of schemes (e.g., MCM) proven secure while modeling components as ideal injections remains when these objects are replaced by PRIOs.

At first glance the notion of a PRIO might appear to be essentially equivalent to that of a pseudorandom oracle. The distinction is somewhat analogous to the difference between PRPs and PRFs. Indeed, random injections and random functions behave similarly up to a birthday-bound, which implies that any PRIO is a good PRO and vice versa. But the more important (and subtle) concern is that the closeness of the definitions might lead one to the conclusion that there are trivial constructions for our mixing steps, utilizing any PRO. However, this would be entirely insufficient for our application because, while a PRO appears injective with high probability, it is *not* necessarily injective by construction. Once we step outside of idealized models we would then have a standard model object that *does not* suffice for the collision-resistance guarantee of Section 3. So for clarity of exposition and analysis, we found it useful to draw a distinction between the two objects.

Building a PRIO that is injective by construction from a blockcipher (modeled as ideal) proves a challenging task. Our object must be publically computable, so no secret keys are allowed. A minimum intuitive security requirement for the object is that the outputs resulting from applying it to two messages that differ in a single bit must appear to have been chosen independently at random, even when adversaries have direct access to the underlying blockcipher. This rules out the straightforward use of existing blockcipher modes of operation, such as CBC, with a public key and fixed IV or even the more complex variable-length enciphering schemes (e.g. [21, 20, 30, 19]).

The TE construction. Our construction utilizes two blockciphers and a trapdoor one-way permutation. Note that in the ideal cipher model one can easily derive two ciphers from a single cipher $\hat{E}$ at the cost of one bit of keying material: $E(K, M) \equiv \hat{E}(1 \,||\, K, M)$ and $E'(K, M) \equiv \hat{E}(0 \,||\, K, M)$. For simplicity then we assume access to two ciphers E: $\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ and E': $\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. The cipher E will be used in a blockcipher mode much like CTR mode encryption. The cipher E' will be utilized to build a function $\mathcal{F}$ for generating *tags* that will be (with high probability) unique to each input message. A message's tag then serves as

algorithm $\mathcal{E}_f(M)$:
$T \leftarrow \mathcal{F}(M)$
$M_1 M_2 \cdots M_m \xleftarrow{n} M$
For $i = 1$ to m do
  $Y_i \leftarrow E(T, \langle i \rangle) \oplus M_i$
$Y_0 \leftarrow f(T)$
Ret $Y_0 \,||\, Y_1 \,||\, \cdots \,||\, Y_m$

algorithm $\mathcal{F}(M)$:
$M_1 M_2 \cdots M_l \xleftarrow{n} \mathsf{PadPF}(M)$
$X_0 \leftarrow IV1$
For $i = 1$ to l do
  $X_i \leftarrow E'(M_i, X_{i-1}) \oplus X_{i-1}$
Ret X_l

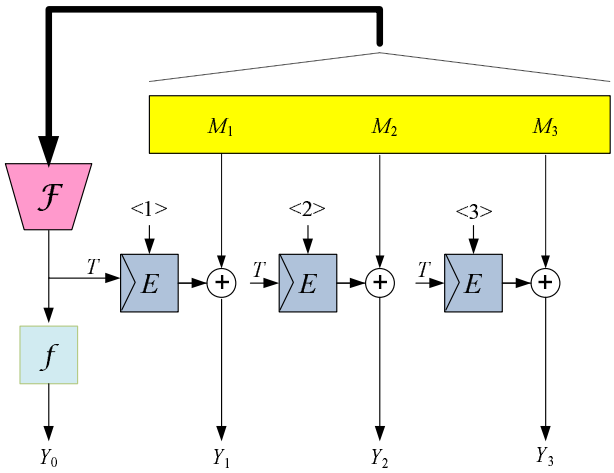

Figure 5: A description of $\mathsf{TE}[E, \mathcal{F}] = (\mathbb{F}_f, \mathcal{E}, \mathcal{M}, \mathcal{M}^+, k)$. **(Top)** Algorithm $\mathcal{E}_f$ and the description of function $\mathcal{F}$. **(Bottom)** A diagram of $\mathcal{E}_f$ applied to a message M for which $|M| = 3n$.

the key for a CTR-mode-like enciphering step. In fact our function $\mathcal{F}$ will realize a blockcipher-based construction of a pseudorandom oracle, originally suggested in [13] and proven secure in [11]. Finally, a trapdoor one-way permutation f is applied to the tag value, the result being the first portion of the output. This step ensures the injectivity of the construction, while the one-wayness "hides" the tag. We will require the trapdoor property in the proof.

Let $\mathcal{M} = \{0,1\}^{\leq L'}$ where $L' = n \cdot 2^{128}$. Let $\mathcal{M}^+ = \mathcal{M} \,||\, \{0,1\}^k$. Let $\mathbb{F}$ be a trapdoor permutation generator. Then we define the cryptographic function $\mathsf{TE}[E, \mathcal{F}] = (\mathbb{F}_f, \mathcal{E}, \mathcal{M}, \mathcal{M}^+, k)$ as follows. The algorithm $\mathbb{F}_f$ runs $(f, f^{-1}) \xleftarrow{\$} \mathbb{F}(1^k)$ and outputs f. The algorithm $\mathcal{E}_f$ and a realization of $\mathcal{F}$ are specified in Figure 5. Used there is a prefix-free padding function PadPF: $\{0,1\}^* \to (\{0,1\}^n)^+$. This means that for any two messages $M, M' \in \{0,1\}^*$ with $|M| \neq |M'|$ the string $\mathsf{PadPF}(M)$ is not a prefix of $\mathsf{PadPF}(M')$. (Such functions are simple, one example is to unambiguously pad M to sequence of $n-1$ bit blocks. Then append a zero to all the blocks except the last, to which a one is appended.) Note that the security of TE relies on adversaries not knowing f^{-1}.[4] If $E \xleftarrow{\$} \mathrm{BC}(k, n)$ and $\mathcal{F} = \mathsf{RF}_{\mathcal{M},k}$, we write $\mathcal{E}^{E,\mathcal{F}}(M)$ to denote computing $\mathcal{E}$ on message M using oracle access to E and $\mathcal{F}$.

The security of TE. First we point out that our realization of $\mathcal{F}$ above is a PRO, based on the proof in [11]. The composability guarantees of the indifferentiability framework established in [23] then allow us to just treat $\mathcal{F}$ as a random oracle. As per the definition in Section 2, we allow the simulator in game PRIO to choose the trapdoor one-way permutation f used in TE. Namely, the

[4] Note also that k might be too small for a secure instance of $\mathbb{F}$, in which case one might have to use $\mathcal{F}$ repeatedly to ensure a full domain point of f is generated.

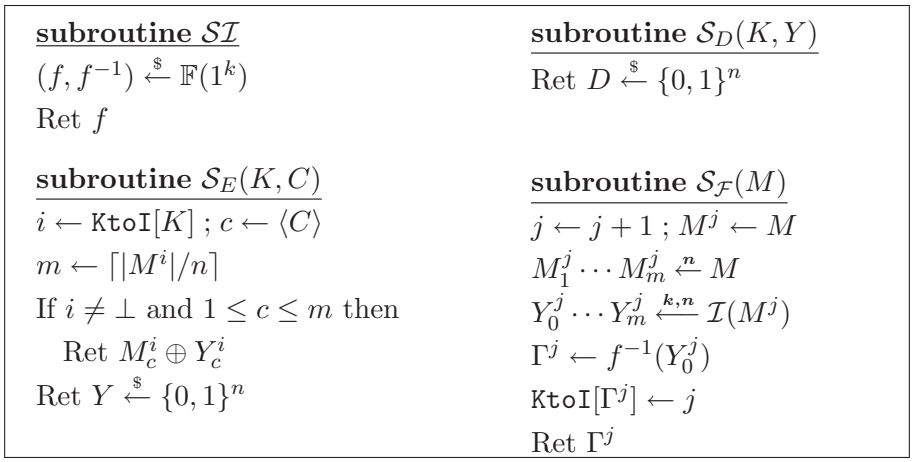

Figure 6: The simulator $\mathcal{S} = (\mathcal{SI}, \mathcal{S}_E, \mathcal{S}_D, \mathcal{S}_{\mathcal{F}})$ used in the proof of Theorem 6.1.

Initialize procedure of $\mathrm{PRIO}_{\mathcal{E},\mathcal{S}}$ (for some simulator $\mathcal{S} = (\mathcal{SI}, \mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$) runs $\mathcal{SI}$ and returns its return value. In game $\mathrm{CONS}_{\mathcal{E}}$, **Initialize** directly runs $(f, f^{-1}) \xleftarrow{\$} \mathbb{F}(1^k)$ and returns f.

Our simulator is shown in Figure 6, where $\mathcal{I} = \mathsf{RI}^*_{\mathcal{M},k}$. Recall that the notation $M^j_1 \cdots M^j_m \xleftarrow{n} M$ for some string M means, informally, to parse M into as many n-bit blocks as possible, putting the remaining bits in a final string M^j_m. Likewise $Y^j_0 \cdots Y^j_m \xleftarrow{k,n} \mathcal{I}(M^j)$ for some string M^j means parse the string returned by $\mathcal{I}$ into a k-bit prefix, as many n-bit blocks as possible, and putting the remaining bits into a final string Y^j_m.

The next theorem captures the main result of this section.

Theorem 6.1 *Fix $k \geq 0$ and $n \geq 0$. Let $E \xleftarrow{\$} \mathrm{BC}(k, n)$ be an ideal cipher with inverse D, let $\mathcal{F} = \mathsf{RF}_{\mathcal{M},k}$, and let $\mathbb{F}$ be a trapdoor permutation generator. Let $\mathsf{TE}[E, \mathcal{F}] = (\mathbb{F}_f, \mathcal{E}, \mathcal{M}, \mathcal{M}^+, k)$ be as specified above. Let $\mathcal{A}$ be an adversary that asks at most (q_0, q_1, q_2, q_3) oracle queries, each of length at most μ bits, and runs in time at most t. Then there exists an adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\mathrm{prio}}_{\mathcal{E},\mathcal{S}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{spowf}}_{\mathbb{F},k}(\mathcal{B}) + \frac{(q_0\sigma + q_1 + q_2)^2}{2^n} + \frac{(q_0 + q_3)^2}{2^k}\,.$$

where $\sigma = \lceil \mu/n \rceil$. Let $q = q_0 + q_1 + q_2 + q_3$. The simulator $\mathcal{S}$, defined in Figure 6, runs in time at most $t_{\mathcal{S}} \leq c\mu(q_3 \mathsf{Time}_{f^{-1}}(1^k) + q \log q)$ for some absolute constant c and makes q_3 queries to its oracle. Adversary $\mathcal{B}$ runs in time $t' \leq t + c'\mu(q_1 \mathsf{Time}_f(1^k) + q \log q)$ for some absolute constant c' and makes at most $(q_0 + q_3, q_3)$ queries to its first oracle and second oracles respectively. ▌

A proof of the theorem is provided in Section 7, here we just provide a brief proof sketch. An adversary is given either oracles implementing the tuple of algorithms $(\mathcal{E}, E, D, \mathcal{F})$ or $(\mathcal{I}, \mathcal{S}_E, \mathcal{S}_D, \mathcal{S}_{\mathcal{F}})$ where $\mathcal{I} = \mathsf{RI}^*_{\mathcal{M},k}$. Recall that D is the oracle implementing the inverse of E. Intuitively the structure of TE ensures that an adversary, attempting to discover information about the tag and via it the random pad created for some message M, must reveal M to the simulator (by querying 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 RO. 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 $\mathcal{S}_E$) 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. Since neither event occurs with high probability, we achieve a bound on the adversary's ability to differentiate the two sets of oracles.

Discussion. One might wonder if we can dispense with the one way permutation. In fact it 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 $T \,||\, Y_1$. 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.

The TE construction is a proof-of-concept: it is the first object to achieve our new goal of being simultaneously constructively injective and indifferentiable from a random injection. On the other hand it has several drawbacks when considering it for practical use. It is length-increasing (outputs are larger than the inputs by at least the number of key bits of the underlying blockcipher). This means that when utilized in MCM the output hash values will be larger compared to the outputs of the provably CR function H. Further, the construction 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, provably collision-resistant function H might not be prohibitive. All this said, the TE construction *does* show that the MCM approach is feasible. We hope that future research will surface improvements.

7 Proof of Theorem 6.1

Overview. Our proof is broken down into several lemmas. Lemma 7.1 shows how to move the $\text{CONS}_{\mathcal{E}}$ game closer to the $\text{PRIO}_{\mathcal{E},\mathcal{S}}$ game, without significant loss. Lemma 7.2 works in the other direction, modifying the $\text{PRIO}_{\mathcal{E},\mathcal{S}}$ game to move it closer to $\text{CONS}_{\mathcal{E}}$. These lemmas culminate in showing that the PRIO security experiment is captured by the difference between two identical-until-bad games G0 and G1. After applying the fundamental lemma of game-playing [4], the rest of the proof involves bounding the probability that bad is set in these games. This last is captured by Lemma 7.3, and involves several more game transitions from G1 to a setting in which one can show that bad is set only if the adversary can invert f on some point. (Recall that in Section 2 we discuss that a one-way function is also a some-point one-way function.)

For the rest of this section, let $k \geq 0$ and $n \geq 0$ be numbers. Let $\mathcal{A}$ be a prio adversary against $\mathsf{TE}[E, \mathcal{F}] = (\mathbb{F}_f, \mathcal{E}, \mathcal{M}, \mathcal{M}^+, k)$ for $E = \mathsf{IC}_{k,n}$, $\mathcal{F} = \mathsf{RF}_{\mathcal{M},k}$. Assume $\mathcal{A}$ makes at most (q_0, q_1, q_2, q_3) queries to its four oracles. Let μ be the maximal length of message queried by $\mathcal{A}$ and define $\sigma = \lceil \mu/n \rceil$. The simulator $\mathcal{S} = (\mathcal{SI}, \mathcal{S}_E, \mathcal{S}_D, \mathcal{S}_F)$ is defined in Figure 6.

We first state the separate lemmas and use them to conclude, then prove each in turn.

Lemma 7.1 $\mathbf{Adv}(\mathcal{A}^{\text{CONS}_{\mathcal{E}}}, \mathcal{A}^{\text{G0}}) \leq (\sigma q_0 + q_1 + q_2)^2 / 2^{n+1}$ □

Lemma 7.2 $\mathbf{Adv}(\mathcal{A}^{\text{G1}}, \mathcal{A}^{\text{PRIO}_{\mathcal{E}}}) \leq (q_0 + q_3)^2 / 2^{k+1}$ □

The above lemmas, combined with the fact that we can apply the fundamental lemma of game-

playing [4] due to G0 and G1 being identical-until-bad, give that

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prio}}_{\mathcal{E},\mathcal{S}}(\mathcal{A}) &\le \mathbf{Adv}(\mathcal{A}^{\mathrm{G0}},\mathcal{A}^{\mathrm{G1}}) + \frac{(\sigma q_0+q_1+q_2)^2}{2^{n+1}} + \frac{(q_0+q_3)^2}{2^{k+1}} \\ &\le \Pr\left[\,\mathcal{A}^{\mathrm{G1}} \text{ sets } \mathsf{bad}\,\right] + \frac{(\sigma q_0+q_1+q_2)^2}{2^{n+1}} + \frac{(q_0+q_3)^2}{2^{k+1}}\,. \qquad (14)\end{aligned}$$

The next lemma captures the bulk of the proof.

Lemma 7.3 *There exists a spowf adversary $\mathcal{B}$ such that*

$$\Pr\left[\,\mathcal{A}^{\mathrm{G1}} \text{ sets } \mathsf{bad}\,\right] \le \mathbf{Adv}^{\mathrm{spowf}}_{\mathbb{F},k}(\mathcal{B}) + \frac{(q_0+q_3)^2}{2^{k+1}}$$

and $\mathcal{B}$ runs in time $t' \le t + c\mu(q_1\mathsf{Time}_{\mathbb{F}}(1^k) + q\log q)$ and makes at most q_0+q_3 queries to its PI oracle and at most q_3 queries to its Inv oracles. □

Combining this last lemma with (14) implies Theorem 6.1.

7.1 Proof of Lemma 7.1

We specify a sequence of games R0 ⟶ R1 ⟶ R2 ⟶ R3 ⟶ G0 such that

$$\mathbf{Adv}(\mathcal{A}^{\mathrm{CONS}_{\mathcal{E}}},\mathcal{A}^{\mathrm{R0}}) = 0 \qquad (15)$$

$$\mathbf{Adv}(\mathcal{A}^{\mathrm{R0}},\mathcal{A}^{\mathrm{R1}}) \le \frac{(\sigma q_0+q_1+q_2)^2}{2^{n+1}} \qquad (16)$$

$$\mathbf{Adv}(\mathcal{A}^{\mathrm{R1}},\mathcal{A}^{\mathrm{R2}}) = \mathbf{Adv}(\mathcal{A}^{\mathrm{R2}},\mathcal{A}^{\mathrm{R3}}) = \mathbf{Adv}(\mathcal{A}^{\mathrm{R3}},\mathcal{A}^{\mathrm{G0}}) = 0 \qquad (17)$$

which together imply the lemma statement. The first four games are shown in Figure 7 and game G0 is shown in Figure 9. In all games we omit explicitly showing an **Initialize** procedure; in every game it just computes $(f, f^{-1}) \xleftarrow{\$} \mathbb{F}$ and returns f. We now discuss each game in turn to justify Equations (15) and (16)

Game R0 implements $\mathrm{CONS}_{\mathcal{E}}$. Procedure $\mathcal{O}_0$ implements $\mathcal{E}$ using an ideal cipher $\mathsf{IC}_{k,n}$ and a table F to lazily build the random function $\mathcal{F}$. Table I is built, but never used in R0 since pointless queries are disallowed. (It will be useful in future games). We have justified (15).

Game R1 is the same as R0 except that we replace $\mathsf{IC}_{k,n}$ with two random functions for each key. The table E is used to implement a random function in place of E. Since $\mathcal{E}$ never uses D (the inverse of E) and pointless queries are disallowed, R1 simply returns random bits for each query to $\mathcal{P}_2$. This procedure is omitted, and will be from all future games since it never changes. A standard birthday-bound argument then justifies (16).

Game R2 changes handling of $\mathcal{P}_3$ queries: instead of directly sampling and recording in F, the oracle now queries $\mathcal{O}_0$ on M to retrieve $Y_0^j \cdots Y_m^j$, and then returns $f^{-1}(Y_0^j)$. The consistency check at the beginning of $\mathcal{O}_0$ ensures that if a previous query to $\mathcal{O}_0$ by $\mathcal{A}$ set $\mathtt{F}[M]$, then the correct value is returned. On the other hand, R2 differs from R1 in that a query to $\mathcal{P}_3$ on M might set several random variables that were not set for such a query in R1 (e.g., values in E). However, the adversary learns nothing about these values until it makes a $\mathcal{O}_0(M)$ or $\mathcal{P}_1$ query, and so choosing them in response to the earlier query is just eager sampling. The change is therefore conservative, and so $\mathbf{Adv}(\mathcal{A}^{\mathrm{R1}},\mathcal{A}^{\mathrm{R2}}) = 0$.

Game R3 makes two conservative changes to R2. First, the consistency checks for selection of T in $\mathcal{O}_0$ are dropped. This check is now redundant because $\mathcal{P}_3$ no longer adds entries to F separately

and so consistency is established by the check against $\texttt{I}$ at the beginning of the $\mathcal{O}_0$ procedure. Second, R3 adds an extra consistency check at the beginning of $\mathcal{P}_1$. This is facilitated by adding the table $\texttt{KtoI}$, which has entries added in $\mathcal{P}_3$. We argue that the modification to $\mathcal{P}_1$ does not change the implemented functionality. If $\texttt{KtoI}[K]$ has a non-bottom entry i and $1 \le c \le \lceil |M^i|/n \rceil$ then the value $M_c^i \oplus Y_c^i = M_c^i \oplus P_c \oplus M_c = P_c$. Here M_c and P_c were defined due to the previous query to $\mathcal{P}_3$ which set $\texttt{KtoI}[K]$ to i. Moreover, $P_c = \texttt{E}[K, C]$, and so the value returned is exactly that which would have been returned in game R2. Thus $\mathbf{Adv}(\mathcal{A}^{\mathrm{R2}}, \mathcal{A}^{\mathrm{R3}}) = 0$.

Game G0 is game R3 but with a flag bad that can be set. The flag does not otherwise affect the execution of the game, and so $\mathbf{Adv}(\mathcal{A}^{\mathrm{R3}}, \mathcal{A}^{\mathrm{G0}}) = 0$. We have justified (17).

7.2 Proof of Lemma 7.2

We specify a sequence of games I0 $\longrightarrow$ I1 $\longrightarrow$ G1 such that

$$\mathbf{Adv}(\mathcal{A}^{\mathrm{I0}}, \mathcal{A}^{\mathrm{PRIO}_{\mathcal{E},\mathcal{S}}}) \;\le\; \frac{(q_0+q_3)^2}{2^{k+1}} \tag{18}$$

$$\mathbf{Adv}(\mathcal{A}^{\mathrm{I0}}, \mathcal{A}^{\mathrm{I1}}) \;=\; \mathbf{Adv}(\mathcal{A}^{\mathrm{I1}}, \mathcal{A}^{\mathrm{G1}}) = 0 \tag{19}$$

which together imply the lemma statement. Games I0 and I1 are shown in Figure 8. Game G1 is shown in Figure 9. Procedure **Initialize** is not shown, in all games it simply generates $(f, f^{-1}) \xleftarrow{\$} \mathbb{F}(1^k)$ and returns f. Also oracle $\mathcal{P}_2$ is not shown, in all games it returns a random string of n bits when queried on any key, message pair. We now discuss each game in turn to justify Equations (18) and (19).

In game I0 procedure $\mathcal{O}_0$ implements a random function while $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ implement the $\mathcal{S}_E$, $\mathcal{S}_D$, and $\mathcal{S}_{\mathcal{F}}$ subroutines of the simulator $\mathcal{S} = (\mathcal{SI}, \mathcal{S}_E, \mathcal{S}_D, \mathcal{S}_{\mathcal{F}})$. Here I0 does not explicitly include $\mathcal{SI}$, instead we allow the procedures access to f^{-1} as if $\mathcal{SI}$ was used. The only distinction, then, between I0 and $\mathrm{PRIO}_{\mathcal{E},\mathcal{S}}$ is that $\mathcal{O}_0$ is a random function in the former and a random injection in the latter. The PRP/PRF switching lemma [4] implies (18), since at most $q_0 + q_3$ invocations of $\mathcal{O}_0$ occur, meaning that many bit strings of length at least $k+1$ bits are sampled.

Game I1 modifies the way game I0 implements $\mathcal{O}_0$. However, the values returned by $\mathcal{O}_0$ in I1 are randomly chosen strings of length $k + |M|$ for any message M queried. To see this, note that T is selected uniformly, and by the permutivity of f this means $f(T)$ inherits this distribution. Each Y_i value is equal to $P_i \oplus M_i$ where P_i is chosen randomly, so each Y_i value is a random bit string. We have justified that $\mathbf{Adv}(\mathcal{A}^{\mathrm{I0}}, \mathcal{A}^{\mathrm{I1}}) = 0$.

Game G1 is game I1 but with a flag bad that might be set. The flag does not modify any other variables, and otherwise the games are identical. We have justified (19).

7.3 Proof of Lemma 7.3

We proceed through a sequence of games G1 $\longrightarrow$ G2 $\longrightarrow$ G3 $\longrightarrow$ G4 $\longrightarrow$ G5 and build from game G5 a spowf adversary $\mathcal{B}$. The games are shown in Figures 9 and 10 and $\mathcal{B}$ is shown in Figure 11.

procedure $\mathcal{O}_0(M)$ * Implements $\mathcal{E}$ * Game R0

If $\mathtt{I}[M] \neq \perp$ then Ret $\mathtt{I}[M]$
$M_1 \cdots M_m \xleftarrow{n} M$
$T \xleftarrow{\$} \{0,1\}^k$; $\Phi \leftarrow f(T)$
If $\mathtt{F}[M] \neq \perp$ then $T \leftarrow \mathtt{F}[M]$
$\mathtt{F}[M] \leftarrow T$
for $i = 1$ to m do
  $P_i \leftarrow \mathsf{IC}_{k,n}(\mathsf{E}, T, \langle i \rangle_n)$
  $Y_i \leftarrow P_i \oplus M_i$
Ret $\mathtt{I}[M] \leftarrow \Phi \,||\, Y_1 \,||\, \cdots \,||\, Y_m$

procedure $\mathcal{P}_1(K, C)$ * Implements E *
Ret $\mathsf{IC}_{k,n}(\mathsf{E}, K, C)$

procedure $\mathcal{P}_2(K, Y)$ * Implements D *
Ret $\mathsf{IC}_{k,n}(\mathsf{D}, K, Y)$

procedure $\mathcal{P}_3(M)$ * Implements $\mathcal{F}$ *
If $\mathtt{F}[M] = \perp$ then $\mathtt{F}[M] \xleftarrow{\$} \{0,1\}^k$
Ret $\mathtt{F}[M]$

procedure $\mathcal{O}_0(M)$ Game R1

If $\mathtt{I}[M] \neq \perp$ then Ret $\mathtt{I}[M]$
$M_1 \cdots M_m \xleftarrow{n} M$
$T \xleftarrow{\$} \{0,1\}^k$; $\Phi \leftarrow f(T)$
If $\mathtt{F}[M] \neq \perp$ then $T \leftarrow \mathtt{F}[M]$
$\mathtt{F}[M] \leftarrow T$
for $i = 1$ to m do
  $P_i \xleftarrow{\$} \{0,1\}^n$ * use E *
  If $\mathtt{E}[T, \langle i \rangle_n] \neq \perp$ then $P_i \leftarrow \mathtt{E}[T, \langle i \rangle_n]$
  $\mathtt{E}[T, \langle i \rangle_n] \leftarrow P_i$
  $Y_i \leftarrow P_i \oplus M_i$
Ret $\mathtt{I}[M] \leftarrow \Phi \,||\, Y_1 \,||\, \cdots \,||\, Y_m$

procedure $\mathcal{P}_1(K, C)$
$U \xleftarrow{\$} \{0,1\}^n$ * use E *
If $\mathtt{E}[K, C] \neq \perp$ then $U \leftarrow \mathtt{E}[K, C]$
Ret $\mathtt{E}[K, C] \leftarrow U$
* $\mathcal{P}_2$ (not shown) returns random bits *

procedure $\mathcal{P}_3(M)$
If $\mathtt{F}[M] = \perp$ then $\mathtt{F}[M] \xleftarrow{\$} \{0,1\}^n$
Ret $\mathtt{F}[M]$

procedure $\mathcal{O}_0(M)$ Game R2

If $\mathtt{I}[M] \neq \perp$ then Ret $\mathtt{I}[M]$
$M_1 \cdots M_m \xleftarrow{n} M$
$T \xleftarrow{\$} \{0,1\}^k$; $\Phi \leftarrow f(t)$
If $\mathtt{F}[M] \neq \perp$ then $T \leftarrow \mathtt{F}[M]$
$\mathtt{F}[M] \leftarrow T$
for $i = 1$ to m do
  $P_i \xleftarrow{\$} \{0,1\}^n$
  If $\mathtt{E}[T, \langle i \rangle_n] \neq \perp$ then $P_i \leftarrow \mathtt{E}[T, \langle i \rangle_n]$
  $\mathtt{E}[T, \langle i \rangle_n] \leftarrow P_i$
  $Y_i \leftarrow P_i \oplus M_i$
Ret $\mathtt{I}[M] \leftarrow \Phi \,||\, Y_1 \,||\, \cdots \,||\, Y_m$

procedure $\mathcal{P}_1(K, C)$
$U \xleftarrow{\$} \{0,1\}^n$
If $\mathtt{E}[K, C] \neq \perp$ then $U \leftarrow \mathtt{E}[K, C]$
Ret $\mathtt{E}[K, C] \leftarrow U$

procedure $\mathcal{P}_3(M)$
$j \leftarrow j + 1$
$Y_0^j \cdots Y_m^j \xleftarrow{k,n} \mathcal{O}_0(M)$ * sample using $\mathcal{O}_0$ *
Ret $\Gamma \leftarrow f^{-1}(Y_0)$

procedure $\mathcal{O}_0(M)$ Game R3

If $\mathtt{I}[M] \neq \perp$ then Ret $\mathtt{I}[M]$
$M_1 \cdots M_m \xleftarrow{n} M$
$T \xleftarrow{\$} \{0,1\}^k$; $\Phi \leftarrow f(t)$ * remove F check *
for $i = 1$ to m do
  $P_i \xleftarrow{\$} \{0,1\}^n$
  If $\mathtt{E}[T, \langle i \rangle_n] \neq \perp$ then $P_i \leftarrow \mathtt{E}[T, \langle i \rangle_n]$
  $\mathtt{E}[T, \langle i \rangle_n] \leftarrow P_i$
  $Y_i \leftarrow P_i \oplus M_i$
Ret $\mathtt{I}[M] \leftarrow \Phi \,||\, Y_1 \,||\, \cdots \,||\, Y_m$

procedure $\mathcal{P}_1(K, C)$
$i \leftarrow \mathtt{KtoI}[K]$; $c \leftarrow \langle C \rangle$; $m \leftarrow \lceil |M^i|/n \rceil$ * add check *
If $i \neq \perp$ and $1 \leq c \leq m$ then
  Ret $M_c^i \oplus Y_c^i$
$U \xleftarrow{\$} \{0,1\}^n$
If $\mathtt{E}[K, C] \neq \perp$ then $U \leftarrow \mathtt{E}[K, C]$
Ret $\mathtt{E}[K, C] \leftarrow U$

procedure $\mathcal{P}_3(M)$
$j \leftarrow j + 1$; $M^j \leftarrow M$
$M_1^j \cdots M_m^j \xleftarrow{n} M$
$Y_0^j \cdots Y_m^j \xleftarrow{k,n} \mathcal{O}_0(M)$
$\Gamma \leftarrow f^{-1}(Y_0^j)$
$\mathtt{KtoI}[\Gamma] \leftarrow j$ * add table entry *
Ret Γ

Figure 7: Games R0, R1, R2, and R3.

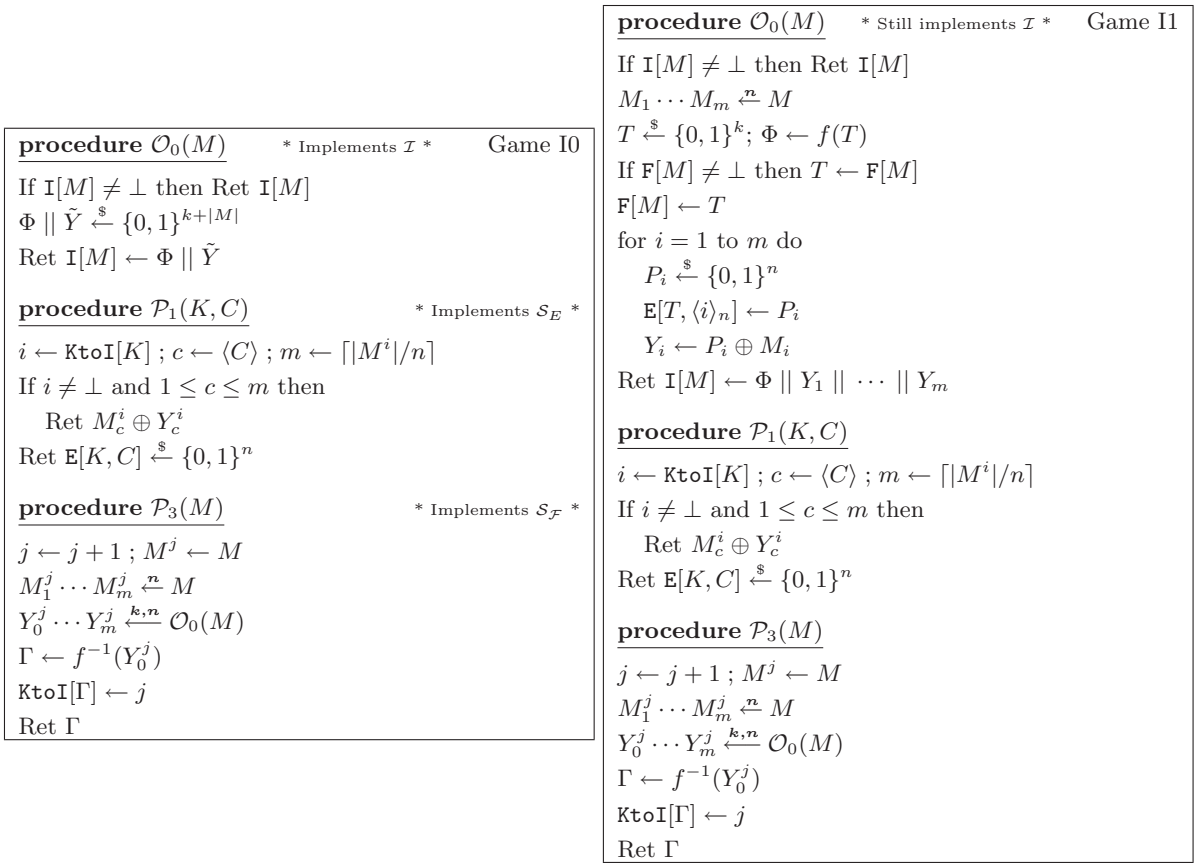

Figure 8: Games I0 and I1. In both cases procedure $\mathcal{P}_2$ is not shown, but simply returns random strings of n bits for any query.

The games and adversary are such that

$$\begin{aligned}
\Pr\left[\mathcal{A}^{\mathrm{G1}} \text{ sets } \mathsf{bad}\right] &= \Pr\left[\mathcal{A}^{\mathrm{G2}} \text{ sets } \mathsf{bad}\right] && (20)\\
&= \Pr\left[\mathcal{A}^{\mathrm{G3}} \text{ sets } \mathsf{bad}\right] && (21)\\
&\leq \Pr\left[\mathcal{A}^{\mathrm{G4}} \text{ sets } \mathsf{bad}\right] + \frac{(q_0+q_3)^2}{2^{k+1}} && (22)\\
&= \Pr\left[\mathcal{A}^{\mathrm{G5}} \text{ sets } \mathsf{bad}\right] + \frac{(q_0+q_3)^2}{2^{k+1}} && (23)\\
&\leq \mathbf{Adv}^{\mathrm{spowf}}_{\mathbb{F},k}(\mathcal{B}) + \frac{(q_0+q_3)^2}{2^{k+1}} && (24)
\end{aligned}$$

These equations (along with the analysis of $\mathcal{B}$'s running time below) imply Lemma 7.3. We discuss each game transition in turn and then $\mathcal{B}$ to justify equations (20) through (24).

Game G2 is the same as G1 except that instead of recording points in $\mathtt{E}$ based on T or K, points are recorded via $f(T)$ and $f(K)$. Since f is a permutation this does not change the implemented functionality, justifying (20). Game G3 (boxed statements are omitted) adds a set $\mathcal{T}$ for recording choices of T in $\mathcal{O}_0$ and a flag $\mathsf{bad2}$ that is set if two values T are chosen to be the same value in the course of the game. The functionality of the oracle is not modified, however, leading to equality (21). Game G4 (boxed statements included) restricts sampling of T to not allow such duplicates.

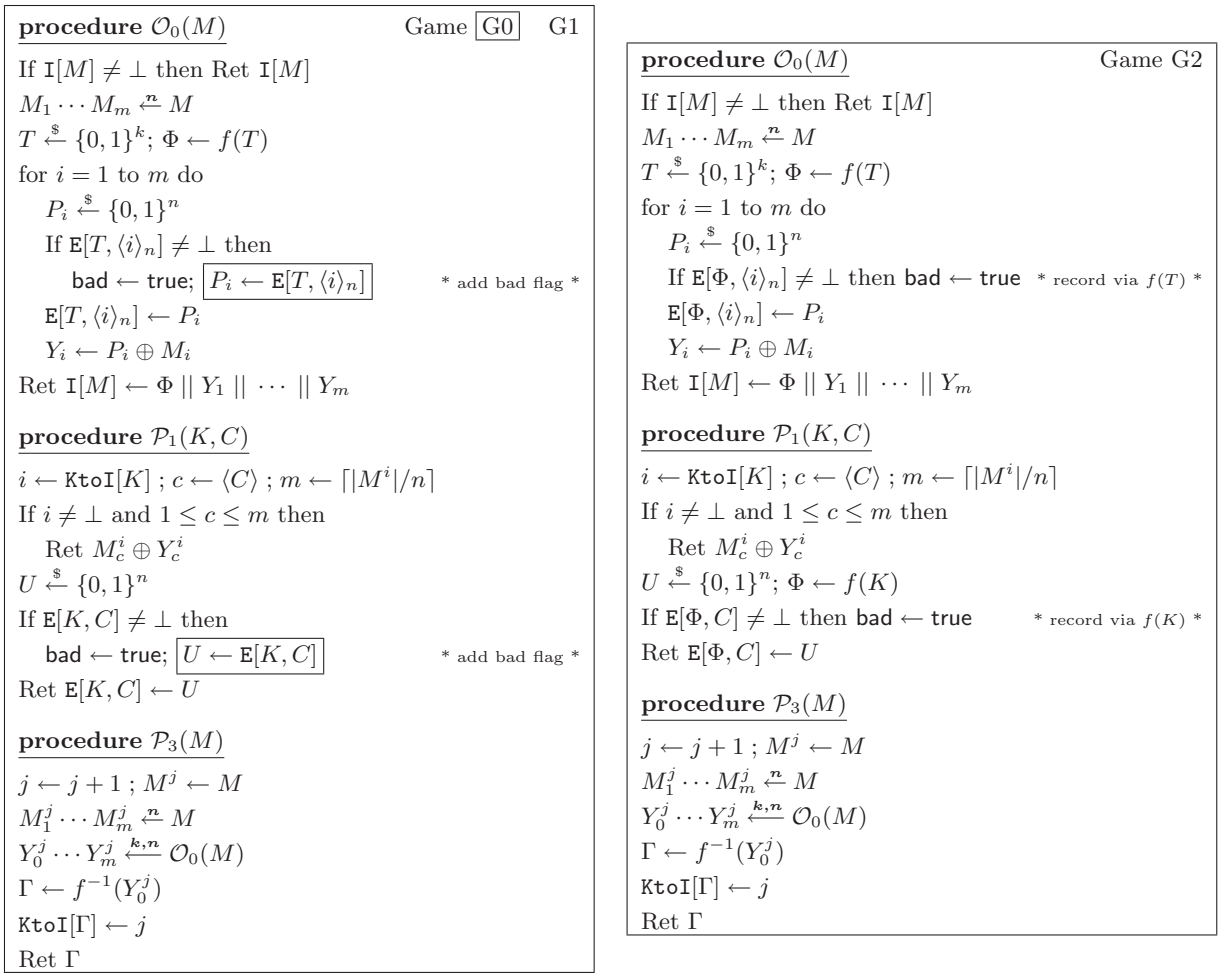

Figure 9: Games G0 (boxed statements included), G1 (boxed statements omitted), and G2.

We have that

$$\Pr\left[\mathcal{A}^{\mathrm{G3}} \text{ sets } \mathsf{bad}\right] - \Pr\left[\mathcal{A}^{\mathrm{G4}} \text{ sets } \mathsf{bad}\right] \leq \Pr\left[\mathcal{A}^{\mathrm{G4}} \text{ sets } \mathsf{bad2}\right] \tag{25}$$

by the fact that G3 and G4 are identical-until-bad2 and the fundamental lemma of game-playing [4]. At most $q_0 + q_3$ values are added to $\mathcal{T}$ in G4. The probability that any pair of such points collides is at most $1/2^k$, so

$$\Pr\left[\mathcal{A}^{\mathrm{G4}} \text{ sets } \mathsf{bad2}\right] \leq \frac{(q_0 + q_3)^2}{2^{k+1}} .$$

Combining the above with (25) justifies (22). Game G5 just samples from $\{0,1\}^k \backslash \mathcal{T}$ directly when choosing T values in $\mathcal{O}_0$. This conservative change implements the same functionality as in G4, which justifies (23).

Adversary $\mathcal{B}$ works just like $\mathcal{A}^{\mathrm{G5}}$ except no initialize procedure is used (instead, $\mathcal{B}$ receives the spowf experiment's choice of f); $\mathcal{O}_0$ is handled using the PI oracle provided to $\mathcal{B}$; and $\mathcal{P}_3$ uses the Inv oracle provided to $\mathcal{B}$ to invert some image points. Several observations are in order about the behavior of $\mathcal{B}$. First, every point queried to Inv necessarily is a value returned by PI since Y_0^j is returned by $\mathcal{O}_0$. Second, if bad is set, then (i) the value $\mathtt{P}[\Phi]$ output by $\mathcal{B}$ is the preimage of an image Φ returned by PI and (ii) $\mathcal{B}$ never queried Φ to Inv. To justify (i) we point out that bad is set

procedure $\mathcal{O}_0(M)$ Game G3 G4

If $\mathtt{I}[M] \neq \perp$ then Ret $\mathtt{I}[M]$
$M_1 \cdots M_m \xleftarrow{n} M$
$T \xleftarrow{\$} \{0,1\}^k$
If $T \in \mathcal{T}$ then * check for duplicate selections *
  $\mathsf{bad2} \leftarrow \mathsf{true}$; $\boxed{T \xleftarrow{\$} \{0,1\}^k \backslash \mathcal{T}}$
$\mathcal{T} \xleftarrow{\cup} T$; $\Phi \leftarrow f(T)$
for $i = 1$ to m do
  $P_i \xleftarrow{\$} \{0,1\}^n$
  If $\mathtt{E}[\Phi, \langle i \rangle_n] \neq \perp$ then $\mathsf{bad} \leftarrow \mathsf{true}$
  $\mathtt{E}[\Phi, \langle i \rangle_n] \leftarrow P_i$
  $Y_i \leftarrow P_i \oplus M_i$
Ret $\mathtt{I}[M] \leftarrow \Phi \,||\, Y_1 \,||\, \cdots \,||\, Y_m$

procedure $\mathcal{P}_1(K, C)$

$i \leftarrow \mathtt{KtoI}[K]$; $c \leftarrow \langle C \rangle$; $m \leftarrow \lceil |M^i|/n \rceil$
If $i \neq \perp$ and $1 \leq c \leq m$ then
  Ret $M_c^i \oplus Y_c^i$
$U \xleftarrow{\$} \{0,1\}^n$; $\Phi \leftarrow f(K)$
If $\mathtt{E}[\Phi, C] \neq \perp$ then $\mathsf{bad} \leftarrow \mathsf{true}$
Ret $\mathtt{E}[\Phi, C] \leftarrow U$

procedure $\mathcal{P}_3(M)$

$j \leftarrow j + 1$; $M^j \leftarrow M$
$M_1^j \cdots M_m^j \xleftarrow{n} M$
$Y_0^j \cdots Y_m^j \xleftarrow{k,n} \mathcal{O}_0(M)$
$\Gamma \leftarrow f^{-1}(Y_0^j)$
$\mathtt{KtoI}[\Gamma] \leftarrow j$
Ret Γ

procedure $\mathcal{O}_0(M)$ Game G5

If $\mathtt{I}[M] \neq \perp$ then Ret $\mathtt{I}[M]$
$M_1 \cdots M_m \xleftarrow{n} M$
$T \xleftarrow{\$} \{0,1\}^k \backslash \mathcal{T}$ * sample directly *
$\mathcal{T} \xleftarrow{\cup} T$; $\Phi \leftarrow f(T)$
for $i = 1$ to m do
  $P_i \xleftarrow{\$} \{0,1\}^n$
  If $\mathtt{E}[\Phi, \langle i \rangle_n] \neq \perp$ then $\mathsf{bad} \leftarrow \mathsf{true}$
  $\mathtt{E}[\Phi, \langle i \rangle_n] \leftarrow P_i$
  $Y_i \leftarrow P_i \oplus M_i$
Ret $\mathtt{I}[M] \leftarrow \Phi \,||\, Y_1 \,||\, \cdots \,||\, Y_m$

procedure $\mathcal{P}_1(K, C)$

$i \leftarrow \mathtt{KtoI}[K]$; $c \leftarrow \langle C \rangle$; $m \leftarrow \lceil |M^i|/n \rceil$
If $i \neq \perp$ and $1 \leq c \leq m$ then
  Ret $M_c^i \oplus Y_c^i$
$U \xleftarrow{\$} \{0,1\}^n$; $\Phi \leftarrow f(K)$
If $\mathtt{E}[\Phi, C] \neq \perp$ then $\mathsf{bad} \leftarrow \mathsf{true}$
Ret $\mathtt{E}[\Phi, C] \leftarrow U$

procedure $\mathcal{P}_3(M)$

$j \leftarrow j + 1$; $M^j \leftarrow M$
$M_1^j \cdots M_m^j \xleftarrow{n} M$
$Y_0^j \cdots Y_m^j \xleftarrow{k,n} \mathcal{O}_0(M)$
$\Gamma \leftarrow f^{-1}(Y_0^j)$
$\mathtt{KtoI}[\Gamma] \leftarrow j$
Ret Γ

Figure 10: Games G3 (boxed statement omitted), G4 (boxed statement included), and G5.

only in the case that a query to $\mathcal{O}_0$ set a point $\mathtt{E}[\Phi, \langle i \rangle_n]$ and a query to $\mathcal{P}_1$ was made with K such that $f(K) = \Phi$. (In particular, the fact that PI samples without replacement means bad cannot be set just due to $\mathcal{O}_0$ queries and the fact that $\mathcal{A}$ cannot make pointless queries means bad cannot be set just due to $\mathcal{P}_1$ queries.) To justify (ii), we note that no value $\mathtt{P}[\Phi]$ is ever recorded after a query $\mathsf{Inv}(Y_0^j)$ for $Y_0^j = \Phi$. This is so because of the check in the second line of $\mathcal{P}_1$. We have justified (24).

The running time of $\mathcal{B}$ is at most the time to run $\mathcal{A}$ plus overhead proportional to $q_1 \mathsf{Time}_f(k) + q \log q$ where $q = q_0 + q_1 + q_2 + q_3$. The $\log q$ factor accounts for the time to implement the tables $\mathtt{P}$, $\mathtt{E}$, and $\mathtt{KtoI}$. Thus, $t' \leq t + c\mu(q_1 \mathsf{Time}_f(k) + q \log q)$ where μ is the longest message queried by $\mathcal{A}$ and c is a small, absolute constant. Moreover, $\mathcal{B}$ makes $q_0 + q_3$ queries to its PI oracle and makes q_3 queries to its Inv oracle.

8 Composability Limitations and Open Problems

Recall that the key benefit of indifferentiability results is the guarantee of composability, as discussed in depth in [23]. For example, a cryptographic scheme $\mathcal{E}$ proven secure when utilizing a (monolithic) random oracle $\mathcal{R}$ remains secure if the random oracle is replaced by a PRO construction C. When

Adversary $\mathcal{B}^{\mathsf{PI},\mathsf{Inv}}(f)$
Run $\mathcal{A}$, answering oracle queries as follows:

query $\mathcal{O}_0(M)$
If $\mathtt{I}[M] \neq \perp$ then Ret $\mathtt{I}[M]$
$M_1 \cdots M_m \xleftarrow{n} M$
$\Phi \leftarrow \mathsf{PI}$ * use image oracle *
for $i = 1$ to m do
  $P_i \xleftarrow{\$} \{0,1\}^n$
  If $\mathtt{E}[\Phi, \langle i \rangle_n] \neq \perp$ then $\mathsf{bad} \leftarrow \mathsf{true}$; Output $\mathtt{P}[\Phi]$
  $\mathtt{E}[\Phi, \langle i \rangle_n] \leftarrow P_i$
  $Y_i \leftarrow P_i \oplus M_i$
Ret $\mathtt{I}[M] \leftarrow \Phi \,||\, Y_1 \,||\, \cdots \,||\, Y_m$

query $\mathcal{P}_1(K, C)$
$i \leftarrow \mathtt{KtoI}[K]$; $c \leftarrow \langle C \rangle$; $m \leftarrow \lceil |M^i|/n \rceil$
If $i \neq \perp$ and $1 \leq c \leq m$ then
  Ret $M^i_c \oplus Y^i_c$
$\Phi \leftarrow f(K)$; $\mathtt{P}[\Phi] \leftarrow K$ * record K *
If $\mathtt{E}[\Phi, C] \neq \perp$ then $\mathsf{bad} \leftarrow \mathsf{true}$; Output $\mathtt{P}[\Phi]$
Ret $\mathtt{E}[\Phi, C] \leftarrow U$

query $\mathcal{P}_3(M)$
$j \leftarrow j + 1$; $M^j \leftarrow M$
$M^j_1 \cdots M^j_m \xleftarrow{n} M$
$Y^j_0 \,||\, \tilde{Y}^j \leftarrow \mathcal{O}_0(M)$
$\Gamma \leftarrow \mathsf{Inv}(Y^j_0)$ * use inverse oracle *
$\mathtt{KtoI}[\Gamma] \leftarrow j$
Ret Γ

When $\mathcal{A}$ halts, output $\perp$

Figure 11: The spowf adversary $\mathcal{B}$ against $\mathbb{F}$.

we say "remains secure" we mean that the existence of an adversary breaking the security of $\mathcal{E}^{\mathcal{R}}$ implies the existence of an adversary that breaks the security of $\mathcal{E}^{C}$. This means we can safely argue about the security of $\mathcal{E}^{C}$ in two steps: show that C is indifferentiable from $\mathcal{R}$ and then that $\mathcal{E}^{\mathcal{R}}$ is secure. Enabling this approach is a significant benefit of simulation-based definitions (the UC framework is another example [8]). Our results also allow for secure composition, but with important (and perhaps subtle) qualifications.

First, we note that both Theorem 3.2 and Theorem 6.1 differ from previous indifferentiability results because they are complexity-theoretic in nature. Specifically, the indifferentiability of MCM from a random oracle (Theorem 3.2) relies on an adversary's inability to find collisions under H. The indifferentiability of TE from a random injection (Theorem 6.1) relies on an adversary's inability to invert the trapdoor permutation f. We must bound the computational power of the adversary in both results, since an unbounded adversary can *always* find collisions against H or invert f. This means that $\mathcal{E}^{\mathsf{MCM}}$, for example, is secure *only* against computationally-bounded adversaries, even if $\mathcal{E}^{\mathcal{R}}$ is information-theoretically secure. This is a problem for random-oracle-based constructions $\mathcal{E}$ that require information-theoretic security (see, e.g. [7]).

Second, Theorem 6.1 relies on a simulator that knows the trapdoor of the one-way permutation (i.e., it gets to control generation of the permutation). Effectively then, instantiating TE requires a trusted party to publish a description of f, which can be considered a common reference string (CRS). We allow the simulator to choose the CRS in the proof. Recent results by Pass and Canetti et al. [6, 9] call into question the (wide) use of such powerful simulators, in that composability of some security properties might be lost. For example, Pass discusses how deniability of non-interactive zero-knowledge proofs (the prover can assert that he never even proved a statement) does not hold if the proof relies on the zero-knowledge simulator choosing the CRS [6]. Indeed interpreting the composability theorem for the indifferentiability framework [23, Thm. 1] in the context of TGen implies that some security properties (e.g., deniability) of constructions using TE will not hold in settings where other parties are allowed to know f.

These subtle nuances of our results lead to a host of provocative open questions. What other properties, beyond deniability, are compromised by the weak composability guarantees of TE? Is it (im)possible to build PRIOs without relying on such strong simulators? Can we strengthen the MCM security result, or find other constructions, that simultaneously are provably CR and yet have information-theoretic indifferentiability from a RO?

Acknowledgments

The authors thank Yevgeniy Dodis for illuminating discussions regarding composability and deniability and the anonymous reviewers for their valuable comments.

References

[1] M. Bellare, A. Boldyreva, and A. Palacio, An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem. *Advances in Cryptology – EUROCRYPT '04*, LNCS vol. 3027, Springer, pp. 171–188, 2004.

[2] M. Bellare and T. Ristenpart. Multi-property-preserving Hash Domain Extension and the EMD Transform. *Advances in Cryptology – ASIACRYPT '06*, LNCS vol. 4284, Springer, pp. 299–314, 2006.

[3] M. Bellare and T. Ristenpart. Hash Functions in the Dedicated-key Setting: Design Choices and MPP Transforms. *International Colloquium on Automata, Languages, and Programming – ICALP '07*, LNCS vol. 4596, Springer, pp. 399–410, 2007.

[4] M. Bellare and P. Rogaway. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. *Advances in Cryptology – EUROCRYPT '06*, LNCS vol. 4004, Springer, pp. 409–426, 2006.

[5] J. Black, P. Rogaway, and T. Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV.. *Advances in Cryptology – CRYPTO '02*, LNCS vol. 2442, Springer, pp. 320–325, 2002.

[6] R. Pass. On deniability in the common reference String and Random Oracle model. *Advances in Cryptology – CRYPTO '03*, LNCS vol. 2729, Springer, pp. 316–337, 2003.

[7] X. Boyen, Y. Dodis, J. Katz, R. Ostrovsky, and A. Smith. Secure Remote Authentication Using Biometric Data. *Advances in Cryptology – EUROCRYPT '05*, LNCS vol. 3494, Springer, pp. 147–163, 2005.

[8] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. *Symposium on Foundations of Computer Science - FOCS '01*, IEEE Computer Society, pp. 136–145, 2001.

[9] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally Composable Protocols with Global Set-up. *Theory of Cryptography - TCC'07*, LNCS vol. 4392, Springer, pp. 61–85, 2007.

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

[11] D. Chang, S. Lee, M. Nandi, and M. Yung, Indifferentiable Security Analysis of Popular Hash Functions with Prefix-Free Padding. *Advances in Cryptology – ASIACRYPT '06*, LNCS vol. 4284, Springer, pp. 283–298, 2006.

[12] S. Contini, A. Lenstra, and R. Steinfeld. VSH, an Efficient and Provable Collision-Resistant Hash Function.. *Advances in Cryptology – EUROCRYPT '06*, LNCS vol. 4004, Springer, pp. 165–182, 2006.

[13] J.S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgard Revisited: How to Construct a Hash Function. *Advances in Cryptology – CRYPTO '05*, LNCS vol. 3621, Springer, pp. 21–39, 2005.

[14] I. Damgård. Collision-free hash functions and public key signature schemes. *Advances in Cryptology – EUROCRYPT '87*, LNCS vol. 304, Springer, pp. 416–427, 1987.

[15] I. Damgård. A design principle for hash functions. *Advances in Cryptology – CRYPTO '89*, LNCS vol. 435, Springer, pp. 416–427, 1989.

[16] Y. Dodis and P. Puniya. Feistel networks made public, and applications. *Advances in Cryptology – EUROCRYPT '07*. LNCS vol. 4515, Springer, pp. 534–554, 2007.

[17] Y. Dodis and P. Puniya. On the relation between the ideal cipher and random oracle models. *Theory of Cryptography Conference – TCC '06*. LNCS vol. 3876, Springer, pp. 184–206, 2006.

[18] S. Goldwasser and S. Micali. Probabilistic Encryption. Journal of Computer and System Sciences, vol. 28, pp. 270–299, 1984.

[19] 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.

[20] S. Halevi and P. Rogaway. A parallelizable enciphering mode. *Topics in Cryptology – CT-RSA 2004*, LNCS vol. 2964, Springer, pp. 292–304, 2004.

[21] S. Halevi and P. Rogaway. A tweakable enciphering mode. *Advances in Cryptology – CRYPTO 2003*, LNCS vol. 2729, Springer, pp. 482–499, 2003.

[22] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. Provably secure FFT hashing. NIST 2nd Cryptographic Hash Function Workshop (2006).

[23] U. Maurer, R. Renner, and C. Holenstein, Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. *Theory of Cryptography Conference – TCC '04*, LNCS vol. 2951, Springer, pp. 21–39, 2004.

[24] R. Merkle. One way hash functions and DES. *Advances in Cryptology – CRYPTO '89*, LNCS vol. 435, Springer, pp. 428–446, 1989.

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

[26] D. Pointcheval. The Composite Discrete Logarithm and Secure Authentication. *Public Key Cryptography – PKC '00*, LNCS vol. 1751, Springer, pp. 113–128, 2000.

[27] B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. *Advances in Cryptology – CRYPTO'93*, LNCS vol. 773, Springer, pp. 368–378, 1994.

[28] M. Rabin. Digital signatures. Foundations of secure computation, R. A. Millo et. al. eds, Academic Press, 1978.

[29] M. Rabin. Digital signatures and public key functions as intractable as factorization. MIT Laboratory for Computer Science Report TR-212, January 1979.

[30] T. Ristenpart and P. Rogaway. How to Enrich the Message Space of a Cipher. *Fast Software Encryption – FSE '07*, LNCS vol. 4593, Springer, pp. 101–118, 2007.

[31] T. Ristenpart and T. Shrimpton. How to Build a Hash Function from any Collision-Resistant Function (full version of this paper). `http://www.cse.ucsd.edu/users/tristenp/`
[32] R. Rivest, A. Shamir, and L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM 21(2): 120–126 (1978).
[33] R. Rivest and Y. Tauman. Improved online/offline signature schemes. *Advances in Cryptology – CRYPTO '01*, LNCS vol. 2139, Springer, pp. 355–367, 2001.
[34] P. Rogaway and T. Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. *Fast Software Encryption – FSE '04*, LNCS vol. 3017, Springer, pp. 371–388, 2004.
[35] RSA Laboratories, RSA PKCS #1 v2.1: RSA Cryptography Standards (2002).
[36] M. Saarinen. Security of VSH in the Real World. *Progress in Cryptology - INDOCRYPT '06*, LNCS vol. 4329, Springer, pp. 95–103, 2006.
[37] X. Wang, Y.L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. *Advances in Cryptology – CRYPTO '05*, LNCS vol. 3621, Springer, pp. 17–36, 2005.
[38] X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. *Advances in Cryptology – EUROCRYPT '05*, LNCS vol. 3494, Springer, pp. 19–35, 2005.
[39] D. Whitfield and M. Hellman. Privacy and Authentication: An Introduction to Cryptography. Proceedings of the IEEE, 67 (1979), pp. 397–427.

A Proof of Lemma 2.1

Let k be a security parameter and $\mathbb{F}$ be trapdoor permutation generator and $\mathcal{A}$ be an spowf adversary against $\mathbb{F}$ that makes exactly q queries to its PI oracle. Then we construct a owf adversary $\mathcal{B}$ against $\mathbb{F}$ that works as shown below.

adversary $\mathcal{B}(f, Y^*)$
$r \xleftarrow{\$} [1\,..\,q]$; $cnt \leftarrow 0$
$\mathcal{D} \leftarrow \emptyset$
Run $\mathcal{A}^{\mathsf{PI},\mathsf{Inv}}$ answering queries by:
  query PI
    $cnt \leftarrow cnt + 1$
    If $cnt = r$ then reply with Y^*
    $X \xleftarrow{\$} \{0,1\}^k \backslash \mathcal{D}$; $Y \leftarrow f(X)$
    If $Y = Y^*$ then output X
    $\mathcal{D} \xleftarrow{\cup} X$; $\mathtt{R}[Y] \leftarrow X$
    reply with Y
  query $\mathsf{Inv}(Y')$
    If $Y' = Y^*$ then output $\perp$
    reply with $\mathtt{R}[Y']$
When $\mathcal{A}$ outputs X^*, output X^*

Note that $\mathcal{A}$'s environment, as simulated by $\mathcal{B}$, is exactly that of the spowf experiment unless $\mathcal{B}$ aborts early (outputs X or $\perp$). Informally, if the choice r is "correct", then $\mathcal{A}$ won't force $\mathcal{B}$ to halt with output $\perp$ (since by the rules of the spowf game, $\mathcal{A}$ cannot query to Inv the image point it will invert) and otherwise $\mathcal{B}$ outputs the preimage of Y^*. A standard argument rigorously gives this, yielding that

$$\mathbf{Adv}^{\mathrm{spowf}}_{\mathbb{F},k}(\mathcal{A}) \leq q \cdot \mathbf{Adv}^{\mathrm{owf}}_{\mathbb{F},k}(\mathcal{B})\,.$$