Cryptography from Compression Functions: The UCE Bridge to the ROM

Mihir Bellare[1], Viet Tung Hoang[1], and Sriram Keelveedhi

Dept. of Computer Science and Engineering, University of California San Diego, USA.

Abstract. This paper suggests and explores the use of UCE security for the task of turning VIL-ROM schemes into FIL-ROM ones. The benefits we offer over indifferentiability, the current leading method for this task, are the ability to handle multi-stage games and greater efficiency. The paradigm consists of (1) Showing that a VIL UCE function can instantiate the VIL RO in the scheme, and (2) Constructing the VIL UCE function given a FIL random oracle. The main technical contributions of the paper are domain extension transforms that implement the second step. Leveraging known results for the first step we automatically obtain FIL-ROM constructions for several primitives whose security notions are underlain by multi-stage games.Our first domain extender exploits indifferentiability, showing that although the latter does not work directly for multi-stage games it can be used indirectly, through UCE, as a tool for this end. Our second domain extender targets performance. It is parallelizable and shown through implementation to provide significant performance gains over indifferentiable domain extenders.

1 Introduction

Two forms of the random oracle model (ROM) of BR [9] have emerged, namely the VIL-ROM and FIL-ROM. In the VIL-ROM, the random oracle, denoted RO, is variable input length (VIL), meaning takes inputs of arbitrary length. In the FIL-ROM, the random oracle, denoted ro, is fixed input length (FIL), meaning only takes inputs of one, particular length. The VIL-ROM is preferable for the design and analysis of ROM schemes and reflects the original view of BR [9] that random oracles would be instantiated by cryptographic hash functions that, like SHA-256, take variable length inputs. However hash functions are built in a very structured way from their underlying compression functions. This lead researchers beginning with Coron, Dodis, Malinaud and Puniya [14] to suggest that it should be the compression function, rather than the hash function, that is treated as "ideal," leading to the FIL-ROM. Indeed, SHA-256 is built from its compression function sha-256 in a way that renders SHA-256 subject to the extension attack, which can lead to attacks when SHA-256 is used to instantiate a VIL random oracle. Treating the compression function (rather than the full hash function) as the ideal object is more reflective of the design goals and intuition of practitioners and leads to better security.

The consensus then is that we should design schemes in the FIL-ROM. The question is how best to do this. One option is to directly design and analyze schemes in this model, but this is difficult and ad hoc. A better option is to provide a construction E^{ro} of a VIL function that can substitute a VIL RO, meaning we would design schemes secure in the VIL-ROM as usual and then automatically replace RO with E^{ro} to obtain security in the FIL-ROM. We refer to such an E as a domain extension construction or domain extender.

For this to work in some broad and useful way, we need a *definition* of some property, call it X, that, if satisfied by E^{ro}, allows the latter to securely replace RO in the VIL-ROM and thus provide security in the FIL-ROM, for some useful and hopefully large set of schemes that are proven secure in the VIL-ROM. The leading proposal for X is "indifferentiability from a random oracle" as defined by Maurer, Renner and Holenstein [19] and advocated by [14].

This paper suggests, and explores, an alternative X. We suggest that X be the notion of UCE (Universal Computational Extractor) security defined by BHK [6]. Our results will show both theoretical and practical benefits of X= UCE over X= indifferentiability in this role. On the theoretical side, UCE allows us to move from the VIL-ROM to the FIL-ROM for primitives whose security is defined via multi-stage games, a setting where indifferentiability fails [23, 15]. On the practical side, we exhibit UCE domain extenders E that are significantly more efficient than known indifferentiability ones, in particular parallelizable to take advantage of modern multi-core machines, our efficiency claims being not just asymptotic but supported by implementations and experiments. Conceived as a way to remove random oracles, UCE now becomes a bridge to better security in the ROM.

Limitations of indifferentiability. While indifferentiability works well in some settings, it has two major limitations. The first is that indifferentiable-from-RO functions do not suffice to securely replace a VIL random oracle for primitives whose security definition is underlain by multi-stage games [23, 15]. This gap is more than academic, for we are seeing the emergence of numerous primitives and security notions of practical importance whose definitions are inherently multi-stage. Examples include Deterministic PKE (D-PKE) [5], Message-Locked Encryption (MLE) [8], and proofs of storage. In each case there are natural, efficient and canonical solutions in the VIL-ROM that we would like to implement in the FIL-ROM, but indifferentiability offers no way to do this.

The second limitation of indifferentiability is performance. Typical indifferentiable domain extenders iterate the compression function sequentially. This means that instantiations are left unable to take advantage of modern multi-core processors to provide performance gains. This reduces the potential for high volume usage and deployment of cryptography based on compression functions.

Our perspective. We conceptualize the goal that motivated the use of indifferentiability as aiming to design an X-secure domain extender —this being a construction E^{ro} that, given the FIL random oracle ro, computes a VIL, X-secure function— for a "good" choice of X, meaning one that allows E^{ro} to securely replace RO in the VIL-ROM for some significant set of applications.

Method	Notions	Performance	Applications
Keyed-Indiff	$\mathsf{UCE}[\mathcal{S}^{\mathrm{srs}}]$ $\mathsf{UCE}[\mathcal{S}^{\mathrm{crs}}]$	About $m/(m-n)$ times the speed of M	All schemes in [6]
AU-then-Hash	$\mathsf{UCE}[\mathcal{S}^{\mathrm{sup}}]$	Parallelizable ~ 0.4 cycles per byte	MLE, key derivation, storage auditing

Fig. 1. Our UCE domain extension constructions and their properties. The second column gives the UCE notion that is achieved. M is the indifferentiable domain extender used in the first construction. The numbers n and m are the key length of the hash function and the input length of the ideal compression function, respectively. Typically, $n = 128$ and $m = 512$.

While X=indifferentiability has been very successful in some domains, it also, as discussed above, has important limitations. We ask if there are alternative definitions X that can overcome these limitations and complement indifferentiability in its role.

The core limitation of indifferentiability is the inability to handle multi-stage games. We suggest that a natural route around this is that X-*security itself be multi-stage*. The particular candidate X we suggest is the UCE notion of [6], which is indeed multi-stage. Our suggested UCE-based paradigm to move schemes from the VIL-ROM to the FIL-ROM has two steps: (1) Show that instantiating the VIL random oracle in the scheme with a VIL UCE function preserves security, and (2) Implement the VIL UCE function as E^{ro} to obtain a FIL-ROM scheme. Prior work has already given us the first step for many constructions: UCE-secure hash functions are shown in [6] to be able to securely instantiate VIL random oracles for diverse multi-stage applications including the important practical ones noted above and all examples of multi-stage schemes listed in [23]. The missing element is UCE domain extenders E for the second step. If we had those, we could immediately harvest the existing results to get FIL-ROM constructions for many multi-stage primitives. The concrete quest that emerges, then, is for UCE domain extenders.

Our results. Our core contribution is two new domain extenders for UCE that together allow us to reach the above goals of security and speed. These are constructions E that take a FIL random oracle ro and return a VIL, keyed function E^{ro} that meets UCE security notions of BHK in the FIL-ROM. (UCE hash functions are keyed, whence the introduction of a key in this setting. Also, UCE is not a monolithic or single security notion, but rather a framework in which one parameterizes notions of security by classes of "sources." Applications rely on different choices of the starting class. The framework is recalled in Section 3. Here we will avoid the details beyond noting for which classes each of our constructions is secure and what this entails for applications.) See Fig. 1 for a summary of the two domain extenders and their properties.

Our first construction is generic, turning any indifferentiable domain extender into a UCE domain extender. Given an indifferentiable domain extender M, we show that the hash family $\mathsf{H}_{hk} = \mathsf{M}^{\mathsf{ro}(hk \,\|\, \cdot)}$ is UCE-secure. The forms of UCE

for which this works are enough to prove security for all schemes listed in [6], for example the EwH D-PKE scheme of [5], or the storage-auditing scheme used in [23] as a counterexample for the failure of the indifferentiability framework in multi-stage settings.

This construction illustrates what we believe is an interesting relation between UCE and indifferentiability. Indifferentiability cannot *directly* yield the applications we have obtained for multi-stage primitives. However, it can be used, in a blackbox way, to create a domain extender that meets a *particular* multi-stage notion of security, namely UCE. Then, exploiting known UCE results, we can obtain FIL-ROM security for *many* multi-stage primitives. Thus our construction shows how to use UCE to leverage indifferentiability to solve a problem that indifferentiability could not solve directly.

While our first construction delivers, we believe, important advances on the theoretical front, its performance is that of the underlying indifferentiable construction. Our second construction targets speed. It follows the Carter-Wegman paradigm [13], first using an almost-universal hash to condense the input, and then running $\mathsf{ro}(K \,\|\, \cdot)$ on the result, where K is the hash key. This gives us highly efficiently, fully parallelizable hash constructions that are not achievable if the target is indifferentiability. In more detail, we show that if F is almost-universal, then the hash family $\mathsf{H}_{hk}(x) = \mathsf{ro}(K \,\|\, \mathsf{F}_{fk}(x))$, with $hk = (fk, K)$, is UCE-secure. The most important application here is the message-locked encryption (MLE) scheme CE of [8]. Due to the space constraint, we leave the proofs of our theorems to the full version [7].

General domain extension. Above we presented the domain extension problem for notion X as being to design E such that E^{ro} is a VIL X-secure function in the FIL-ROM. More generally, the problem is to design E such that if $\overline{\mathsf{H}}$ is a FIL X-secure function then $\mathsf{E}^{\overline{\mathsf{H}}}$ is a VIL X-secure function. Here $\overline{\mathsf{H}}$ can be a FIL-ROM function, and thus the prior formulation is the special case $\overline{\mathsf{H}}_{hk}(\cdot) = \mathsf{ro}(hk\|\cdot)$. Our first construction discussed above generalizes to solve this problem, letting $\mathsf{H}_{hk} = \mathsf{M}^{\overline{\mathsf{H}}(hk \,\|\, \cdot)}$ where M, as before, is an indifferentiable domain extender. Setting $\overline{\mathsf{H}}_{hk}(\cdot) = \mathsf{ro}(hk\|\cdot)$ recovers the result stated above. The generalization however yields something new, namely a *standard model* domain extender for UCE. This follows by letting $\overline{\mathsf{H}}$ be a standard model FIL UCE function. This is interesting because it shows that indifferentiability, which so far has been a ROM notion and tool, can be leveraged to get results purely in the standard model.

Instantiation and experimental results. We give a very fast instantiation of F based on reduced-round AES and polynomial-based evaluation. Our construction makes use of the fact that four-round AES, with the four subkeys chosen uniformly and independently, is an almost-xor-universal hash [18]. We stress that our universal hashing construction is unconditional, making no assumption on AES. This leads to a highly efficient, parallelizable UCE-secure hash FastHash. Our experiments show that even in the sequential setting, FastHash is about 5.3 times faster than SHA-256. When parallelism is employed, FastHash achieves a

much better speedup, about 24 times faster than SHA-256. Finally, we demonstrate the utility of FastHash by giving an extremely fast MLE scheme.

Related work. Mittelbach [21] defines restrictions on a multi-stage game so that the indifferentiability composition theorem still holds for a subclass of indifferentiable domain extenders called iterative domain extenders, and is thereby able to show that the latter suffice for applications like D-PKE and MLE. He also shows that if M is an iterative domain extender then M^{ro} is UCE-secure. In comparison, our first construction is more general in the following ways: It is able to use any indifferentiable domain extender, and as a result our applications are able to use a broader class of domain extenders; it turns any FIL UCE function into a VIL one; it works both in the standard model and the ROM. On the other hand, Mittelbach's construction is about $m/(m-n)$ times faster than ours, where m is the input length of the compression function, and n is the key length.

Dodis, Ristenpart, and Shrimpton [16] define preimage-awareness (PrA) as a strengthening of collision resistance and show that the plain Merkle-Damgård is a PrA extender. PrA can also be used in multi-stage games: Ristenpart, Shacham, and Shrimpton [23] show how to compose a PrA-secure hash with a FIL RO to achieve D-PKE.

Some versions of UCE are shown by [12] to be unachievable in the standard mode if indistinguishability obfuscation for all circuits exists, but most of the applications in [6] only need weaker versions of UCE where our domain extenders work but the attacks in [12] do not. All versions of UCE in [6] are shown by the latter to be achievable in the VIL-ROM, so our domain extenders achieve all the applications in the FIL-ROM.

2 Preliminaries

Concrete security bounds are important for applications. However, notions in the current domain, involving simulators and multiple conditions and adversaries, are complex. The result is that when theorems are stated purely concretely, it is hard to understand the (much more simple) conceptual import. We will try to achieve the "best of both worlds." We formulate definitions asymptotically. The first cut theorem statements are asymptotic so that one can quickly see the core implication and result. This is followed by a concrete statement with bounds.

Notation. By $\lambda \in \mathbb{N}$ we denote the security parameter. If $n \in \mathbb{N}$ then 1^n denotes its unary representation. We denote the size of a finite set X by $|X|$, the number of coordinates of a vector $\mathbf{x}$ by $|\mathbf{x}|$, and the length of a string $x \in \{0,1\}^*$ by $|x|$. We let ε denote the empty string. If x is a string then $x[i]$ is its i-th bit and $x[1,\ell] = x[1]\ldots x[\ell]$. By $x\|y$ we denote the concatenation of strings x, y. If X is a finite set, we let $x \xleftarrow{\$} X$ denote picking an element of X uniformly at random and assigning it to x. Algorithms may be randomized unless otherwise indicated. Running time is worst case. "PT" stands for "polynomial-time," whether for randomized algorithms or deterministic ones. If A is an algorithm, we

let $y \leftarrow A(x_1, \ldots; r)$ denote running A with random coins r on inputs $x_1, \ldots$ and assigning the output to y. We let $y \leftarrow\!\!\$\ A(x_1, \ldots)$ be the resulting of picking r at random and letting $y \leftarrow A(x_1, \ldots; r)$. We let $[A(x_1, \ldots)]$ denote the set of all possible outputs of A when invoked with inputs $x_1, \ldots$. We say that $f : \mathbb{N} \to \mathbb{R}$ is negligible if for every positive polynomial p, there exists $n_p \in \mathbb{N}$ such that $f(n) < 1/p(n)$ for all $n > n_p$.

GAMES. We use the code based game playing framework of [10]. (See Fig. 3 for an example.) By $\mathrm{G}^{A_1, A_2, \cdots}(\lambda) \Rightarrow y$ we denote the event that the execution of game G with adversaries $A_1, A_2, \ldots$ and security parameter λ results in output y. We abbreviate $\mathrm{G}^{A_1, A_2, \cdots}(\lambda) \Rightarrow \mathsf{true}$ by $\mathrm{G}^{A_1, A_2, \cdots}(\lambda)$, the occurrence of this event meaning that $A_1, A_2, \ldots$ win the game.

For concrete security assessments, let the *number of queries* of A to an oracle Proc be the function $\mathbf{Q}_A^{\mathsf{Proc}}$ that on input λ returns the maximum number of queries that A makes to Proc when executed with security parameter λ, the maximum over all coins and all possible replies to queries to all oracles of A. Time assessments are simplified by the convention that running time is that of the game rather than merely the adversary, and we let $\mathbf{T}(\mathrm{G}^{A_1, A_2, \cdots})$ denote the function of λ that returns the maximum execution time of game G with adversaries $A_1, A_2, \ldots$ and security parameter λ, the maximum over all coins, and the time being all inclusive, meaning the time taken by game procedures to compute replies is included.

RANDOM ORACLES. A random oracle $\mathsf{RO} : U \to \{0,1\}^n$ is a procedure that maintains a table H, initially empty, and is defined by

$\underline{\mathsf{RO}(x)}$
If $H[x] \neq \perp$ then $H[x] \leftarrow\!\!\$\ \{0,1\}^n$; Return $H[x]$

We say that RO is variable-input length (VIL) if $U = \{0,1\}^*$ and fixed-input length (FIL) if there is $m \in \mathbb{N}$ such that $U = \{0,1\}^m$. Formally, any random oracle referred to in a game should appear explicitly in the game as a procedure defined as above, but for the same of brevity of game descriptions, we omit writing it explicitly, instead only indicating the domain and range of each random oracle. By convention, RO indicates a VIL random oracle, and ro a FIL random oracle.

3 UCE framework

The Universal Computational Extractor (UCE) framework of BHK [6] is intended to define security notions for families of hash functions in the standard model, but BHK also lift this to the ROM to show its achievability there. We use the latter with the random oracle being FIL. We note that the standard-model definition is the special case where parties and algorithms make no queries to the random oracle.

BHK first give a single-key version of the definition and then extend it to a multi-key one. We will work directly with the multi-key version, calling it UCE rather than mUCE as in [6].

FUNCTION FAMILIES. Our syntax for function families follows [6], in particular allowing variable output lengths. A family of functions H specifies the following. On input the unary representation 1^λ of the security parameter $\lambda \in \mathbb{N}$, key generation algorithm $\mathsf{H.Kg}$ returns a key $hk \in \{0,1\}^{\mathsf{H.kl}(\lambda)}$, where $\mathsf{H.kl}\colon \mathbb{N} \to \mathbb{N}$ is the keylength function associated to H. The deterministic, PT evaluation algorithm $\mathsf{H.Ev}$ takes 1^λ, a key $hk \in [\mathsf{H.Kg}(1^\lambda)]$, an input $x \in \{0,1\}^*$ with $|x| \in \mathsf{H.IL}(\lambda)$, and a unary encoding 1^ℓ of an output length $\ell \in \mathsf{H.OL}(\lambda)$ to return $\mathsf{H.Ev}(1^\lambda, hk, x, 1^\ell) \in \{0,1\}^\ell$. Here $\mathsf{H.IL}$ is the input-length function associated to H, so that $\mathsf{H.IL}(\lambda) \subseteq \mathbb{N}$ is the set of allowed input lengths, and similarly $\mathsf{H.OL}$ is the output-length function associated to H, so that $\mathsf{H.OL}(\lambda) \subseteq \mathbb{N}$ is the set of allowed output lengths. The latter allows us to cover functions of variable output length. If H has fixed input length then let $\mathsf{H.il}$ denote the function such that $\mathsf{H.IL}(\lambda) = \{\mathsf{H.il}(\lambda)\}$ for every $\lambda \in \mathbb{N}$. If H has fixed output length, define $\mathsf{H.ol}$ likewise. In the ROM, we allow $\mathsf{H.Ev}$ access to a FIL random oracle denoted ro. We write $\mathsf{H.Ev}^{\mathsf{ro}}$ to indicate explicitly that $\mathsf{H.Ev}$ needs access to a FIL random oracle ro.

FRAMEWORK. Let H be a family of functions. Let S be an adversary called the *source* and D an adversary called the *distinguisher*. We associate to them and H the game $\mathrm{UCE}^{S,D}_{\mathsf{H}}(\lambda)$ in the left panel of Fig. 2. Initially, the source specifies a unary-encoded integer $n \geq 1$ to indicate the number of hash keys that it wants to use. The game then chooses a secret vector $\mathbf{hk}$ of n uniformly random hash keys and grants the source access to an oracle Hash. We require that any query $(x, 1^\ell, i)$ made to this oracle satisfy $|x| \in \mathsf{H.IL}(\lambda)$, $\ell \in \mathsf{H.OL}(\lambda)$ and $i \in \{1, \ldots, n\}$. When the challenge bit b is 1 (the "real" case) the oracle responds via $\mathsf{H.Ev}$ under $\mathbf{hk}[i]$. When $b = 0$ (the "random" case) it responds via the ith random-oracle procedure. The source then leaks a string L to its accomplice distinguisher. The latter *does* get the keys $\mathbf{hk}$ as input and must now return its guess $b' \in \{0,1\}$ for b. The game returns true iff $b' = b$, and the uce-advantage of (S, D) is defined for $\lambda \in \mathbb{N}$ via

$$\mathsf{Adv}^{\mathsf{uce}}_{\mathsf{H},S,D}(\lambda) = 2\Pr[\mathrm{UCE}^{S,D}_{\mathsf{H}}(\lambda)] - 1 \ .$$

If $\mathcal{S}$ is a class (set) of sources, we say that H is $\mathsf{UCE}[\mathcal{S}]$-secure if $\mathsf{Adv}^{\mathsf{uce}}_{\mathsf{H},S,D}(\cdot)$ is negligible for all sources $S \in \mathcal{S}$ and all PT distinguishers D. Trivial attacks from [6] show that $\mathsf{UCE}[\mathcal{S}]$-security is not achievable if $\mathcal{S}$ is the class of all PT sources. To obtain meaningful notions of security, BHK [6] impose restrictions on the source. There are many ways to do this; below we'll focus on what they call unpredictable and reset-secure sources. To discuss the concrete security of constructions it will be useful to say that S is a N-key source if we always have $n \leq N(\lambda)$ when $(1^n, t) \leftarrow\!\!\$\ S(1^\lambda, \varepsilon)$.

UNPREDICTABLE SOURCES. A source is unpredictable if it is hard to guess the source's Hash queries even given the leakage, in the *random case* of UCE game. Formally, let S be a source and P an adversary called a predictor. Consider game $\mathrm{Pred}^P_S(\lambda)$ in the middle panel of Fig. 2 associated to S, P. Given 1^n and the leakage, the predictor outputs a set Q'. The predictor wins if Q' contains a

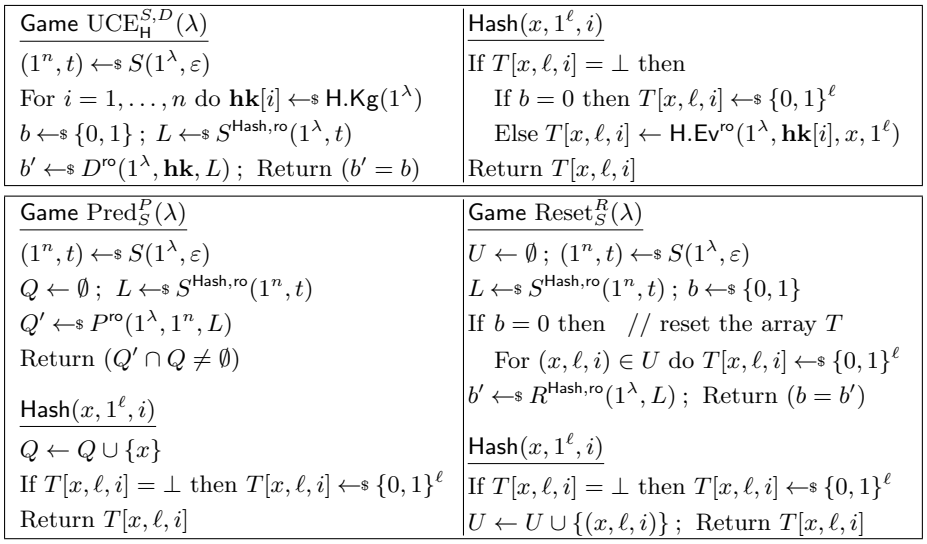

Fig. 2. Games UCE **(top),** Pred **(bottom left), and** Reset **(bottom right) to define UCE security.** Here $\mathsf{ro} : \{0,1\}^{\mathsf{ro.il}(\lambda)} \to \{0,1\}^{\mathsf{ro.ol}(\lambda)}$ is a random oracle.

Hash-query of the source. For $\lambda \in \mathbb{N}$ we let

$$\mathsf{Adv}^{\mathsf{pred}}_{S,P}(\lambda) = \Pr[\mathrm{Pred}^P_S(\lambda)] \ .$$

We require that the size of Q', as well as the number of queries that P makes to ro, be bounded by a polynomial (allowed to depend on P) in λ. We say that S is computationally (respectively, statistically) unpredictable if $\mathsf{Adv}^{\mathsf{pred}}_{S,P}(\cdot)$ is negligible for all PT (respectively, all, even computationally unbounded) predictors P. We let $\mathcal{S}^{\mathrm{cup}}$ be the class of computationally unpredictable PT sources, and $\mathcal{S}^{\mathrm{sup}}$ the class of statistically unpredictable PT sources. The corresponding security notions for H are $\mathsf{UCE}[\mathcal{S}^{\mathrm{cup}}]$ and $\mathsf{UCE}[\mathcal{S}^{\mathrm{sup}}]$.

Reset-secure sources. We recall the second restriction on sources from [6], called reset security. Let S be a source and R an adversary called a reset adversary. The source again is executed with its Hash being a random oracle. The reset adversary is either given access to the same random oracle or to an *independent* one. The requirement is that it should not be able to tell which. Formally, consider game $\mathrm{Reset}^R_S(\lambda)$ at the right panel of Fig. 2 associated to S, R. For $\lambda \in \mathbb{N}$ we let

$$\mathsf{Adv}^{\mathsf{reset}}_{S,R}(\lambda) = 2\Pr[\mathrm{Reset}^R_S(\lambda)] - 1 \ .$$

We require that the number of queries that P makes to Hash and ro be bounded by a polynomial (allowed to depend on R) in λ. We say S is computationally (respectively, statistically) reset-secure if $\mathsf{Adv}^{\mathsf{reset}}_{S,R}(\cdot)$ is negligible for all PT (respectively, all, even computationally unbounded) reset adversaries R. We let $\mathcal{S}^{\mathrm{crs}}$ be the class of all PT computationally reset-secure sources, and $\mathcal{S}^{\mathrm{srs}}$ the class of

$\underline{\mathsf{Game}\ \mathrm{Indiff}^A_{\mathsf{M},\overline{\mathsf{M}}}(\lambda)}$	$\underline{\mathsf{Func}(x)}$	$\underline{\mathsf{Prim}(x)}$
$b \leftarrow\!\!\$\ \{0,1\}$; $\mathrm{st} \leftarrow \varepsilon$	If $b = 1$ then return $\mathsf{M}^{\mathsf{ro}}(1^\lambda, x)$	If $b = 1$ then return $\mathsf{ro}(x)$
$b' \leftarrow\!\!\$\ A^{\mathsf{Prim},\mathsf{Func}}(1^\lambda)$	Else return $\mathsf{RO}(x)$	$(y, \mathrm{st}) \leftarrow\!\!\$\ \overline{\mathsf{M}}^{\mathsf{RO}}(1^\lambda, \mathrm{st}, x)$
Return $(b = b')$		Return y

Fig. 3. Game Indiff **defining indifferentiability.** Here $\mathsf{RO} : \{0,1\}^* \to \{0,1\}^{\mathsf{M.fol}(\lambda)}$ and $\mathsf{ro} : \{0,1\}^{\mathsf{M.pil}(\lambda)} \to \{0,1\}^{\mathsf{M.pol}(\lambda)}$ are random oracles.

all PT statistically reset-secure sources. The corresponding security notions for H are $\mathsf{UCE}[\mathcal{S}^{\mathrm{crs}}]$ and $\mathsf{UCE}[\mathcal{S}^{\mathrm{srs}}]$.

Relations and achievability. Reset security is a relaxation of unpredictability. In particular BHK [6] show that $\mathsf{UCE}[\mathcal{S}^{\mathrm{crs}}]$-security of H implies $\mathsf{UCE}[\mathcal{S}^{\mathrm{cup}}]$-security of H and $\mathsf{UCE}[\mathcal{S}^{\mathrm{srs}}]$-security of H implies $\mathsf{UCE}[\mathcal{S}^{\mathrm{sup}}]$-security of H. The converses are not necessarily true. BFM [12] show that if indistinguishability obfuscation for all circuits is possible then $\mathsf{UCE}[\mathcal{S}^{\mathrm{crs}}]$-security is not achievable in the standard model. In the ROM however BHK [6] show that both $\mathsf{UCE}[\mathcal{S}^{\mathrm{crs}}]$-security and $\mathsf{UCE}[\mathcal{S}^{\mathrm{srs}}]$-security are achievable.

4 UCE from indifferentiability

We first review necessary definitions of the indifferentiability framework [19].

Indifferentiability. We consider an algorithm M that, given a FIL random oracle ro, attempts to have input-output behavior approximating that of a VIL random oracle. Indifferentiability provides one definition of what it means for M to succeed at this task. Consider game $\mathrm{Indiff}^A_{\mathsf{M},\overline{\mathsf{M}}}(\lambda)$ of Fig. 3 associated to M, an algorithm $\overline{\mathsf{M}}$ called a simulator, and an adversary A. In the first world ($b = 1$), oracle Prim implements the FIL random oracle ro while oracle Func implements the construction, namely M^{ro}, that aims to approximate a VIL random oracle. In the second world ($b = 0$), oracle Func implements a true VIL random oracle RO while replies to Prim queries are determined by the simulator that itself has access to RO. The simulator is stateful, its state st being maintained by the game. The input x to M has arbitrary length, the oracle provided to M maps $\mathsf{M.pil}(\lambda)$-bit inputs to $\mathsf{M.pol}(\lambda)$-bit outputs, and M returns outputs of length $\mathsf{M.fol}(\lambda)$, where $\mathsf{M.pil}, \mathsf{M.pol}, \mathsf{M.fol} : \mathbb{N} \to \mathbb{N}$ are functions associated to M called the input-length of M's primitive, output-length of M's primitive, and output-length of M's functionality, respectively. For $\lambda \in \mathbb{N}$ we let

$$\mathsf{Adv}^{\mathsf{indiff}}_{\mathsf{M},\overline{\mathsf{M}},A}(\lambda) = 2\Pr[\mathrm{Indiff}^A_{\mathsf{M},\overline{\mathsf{M}}}(\lambda)] - 1 \ .$$

We require that the number of queries that A makes to its oracles be bounded by a polynomial (allowed to depend on A) in λ. Then we say that M is a *pseudorandom oracle* (PRO) if there is a PT simulator $\overline{\mathsf{M}}$ such that $\mathsf{Adv}^{\mathsf{indiff}}_{\mathsf{M},\overline{\mathsf{M}},A}(\cdot)$ is negligible for every (even computationally unbounded) adversary A.

For concrete security assessments we let $Q_{\overline{\mathsf{M}},q}$ be the function that on input λ returns the maximum, over all $x_1, \ldots, x_q \in \{0,1\}^{\mathsf{M.pil}(\lambda)}$, of the total number of

oracle queries that $\overline{\mathsf{M}}$ makes when run sequentially on inputs $x_1, \ldots, x_q$, starting from state ε. Also let $T_{\overline{\mathsf{M}},q}$ be the function that on input λ returns the maximum, over all $x_1, \ldots, x_q \in \{0,1\}^{\mathsf{M.pil}(\lambda)}$, of the total running time of $\overline{\mathsf{M}}$ when run sequentially on inputs $x_1, \ldots, x_q$, starting from state ε, the time for an oracle query being taken as linear in the length of the query and reply.

THE Keyed-Indiff EXTENDER. Let $\overline{\mathsf{H}}$ be a FIL function family that is $\mathsf{UCE}[\mathcal{S}^{\mathrm{xxx}}]$-secure for some xxx. We want to build a VIL family of functions H that is also $\mathsf{UCE}[\mathcal{S}^{\mathrm{xxx}}]$-secure. Our construction uses as a tool any PRO M with $\mathsf{M.pil} = \overline{\mathsf{H}}.\mathsf{il}$ and $\mathsf{M.pol} = \overline{\mathsf{H}}.\mathsf{ol}$. We associate to M and $\overline{\mathsf{H}}$ the family of functions $\mathsf{H} = \mathsf{Keyed\text{-}Indiff}[\mathsf{M}, \overline{\mathsf{H}}]$ defined as follows. We let $\mathsf{H.IL} = \mathbb{N}$, meaning H is VIL. The output length of H is $\mathsf{H.ol} = \mathsf{M.fol}$. We let $\mathsf{H.Kg} = \overline{\mathsf{H}}.\mathsf{Kg}$, meaning keys for H are the same as for $\overline{\mathsf{H}}$. Finally for any $\lambda \in \mathbb{N}$, any $hk \in [\mathsf{H.Kg}(1^\lambda)]$ and any $x \in \{0,1\}^*$ we let

$$\mathsf{H.Ev}^{\mathsf{ro}}(1^\lambda, hk, x, 1^{\mathsf{H.ol}(\lambda)}) = \mathsf{M}^{\overline{\mathsf{H}}.\mathsf{Ev}^{\mathsf{ro}}(1^\lambda, hk, \cdot, 1^{\overline{\mathsf{H}}.\mathsf{ol}(\lambda)})}(1^\lambda, x)\ . \tag{1}$$

This needs some explanation. Begin by ignoring ro, so that we are looking at a standard-model construction. Recall that M takes an oracle mapping $\{0,1\}^{\mathsf{M.pil}(\lambda)}$ to $\{0,1\}^{\mathsf{M.pol}(\lambda)}$. In the indifferentiability setting, this is a random oracle. Our construction however does something different. It implements M's oracle via the given $\mathsf{UCE}[\mathcal{S}^{\mathrm{xxx}}]$-secure family $\overline{\mathsf{H}}$. The key hk is held fixed. Our claim will be that H is itself $\mathsf{UCE}[\mathcal{S}^{\mathrm{xxx}}]$-secure for $\mathrm{xxx} \in \{\mathrm{crs}, \mathrm{srs}\}$. Something we consider interesting is that this result is entirely standard model, yet uses ROM theory, in the form of a PRO, for the construction and proof. Finally the ro in the construction simply reflects that the result lifts to the ROM. In case $\overline{\mathsf{H}}$ was a ROM family of functions, H will be as well. This extension, together with known applications of $\mathsf{UCE}[\mathcal{S}^{\mathrm{xxx}}]$-security, allow us to implement in the FIL-ROM many constructions given in the VIL-ROM.

RESULT. We view $\mathsf{Keyed\text{-}Indiff}[\mathsf{M}, \cdot]$ as a domain extension transform taking a FIL family $\overline{\mathsf{H}}$ and returning a VIL family $\mathsf{H} = \mathsf{Keyed\text{-}Indiff}[\mathsf{M}, \overline{\mathsf{H}}]$. The following says that this transform preserves $\mathsf{UCE}[\mathcal{S}^{\mathrm{xxx}}]$-security for $\mathrm{xxx} \in \{\mathrm{crs}, \mathrm{srs}\}$.

Theorem 1. *Let $\overline{\mathsf{H}}$ be a hash function family. Let M be a PRO such that $\mathsf{M.pil} = \overline{\mathsf{H}}.\mathsf{il}$ and $\mathsf{M.pol} = \overline{\mathsf{H}}.\mathsf{ol}$. Let $\mathsf{H} = \mathsf{Keyed\text{-}Indiff}[\mathsf{M}, \overline{\mathsf{H}}]$. Let $\mathrm{xxx} \in \{\mathrm{crs}, \mathrm{srs}\}$.*

Asymptotic result: If $\overline{\mathsf{H}}$ is $\mathsf{UCE}[\mathcal{S}^{\mathrm{xxx}}]$-secure then so is H.

Concrete result: Let $\overline{\mathsf{M}}$ be a simulator for M. Let S be an N-key source, D a distinguisher and $\overline{R}$ a reset adversary. Then we construct an N-key source $\overline{S}$, indifferentiability adversaries A, B and a reset adversary R such that

$$\mathsf{Adv}^{\mathsf{uce}}_{\mathsf{H},S,D}(\lambda) \le \mathsf{Adv}^{\mathsf{uce}}_{\overline{\mathsf{H}},\overline{S},D}(\lambda) + N(\lambda) \cdot \mathsf{Adv}^{\mathsf{indiff}}_{\mathsf{M},\overline{\mathsf{M}},A}(\lambda) \tag{2}$$

$$\mathsf{Adv}^{\mathsf{reset}}_{\overline{S},\overline{R}}(\lambda) \le \mathsf{Adv}^{\mathsf{reset}}_{S,R}(\lambda) + 3N(\lambda) \cdot \mathsf{Adv}^{\mathsf{indiff}}_{\mathsf{M},\overline{\mathsf{M}},B}(\lambda) \tag{3}$$

for all $\lambda \in \mathbb{N}$. Furthermore:

$\mathbf{Q}_A^{\mathsf{Prim}} = 0$; $\mathbf{Q}_A^{\mathsf{Func}} = \mathbf{Q}_B^{\mathsf{Func}} = \mathbf{Q}_S^{\mathsf{Hash}}$; $\mathbf{Q}_B^{\mathsf{Prim}} = \mathbf{Q}_{\overline{R}}^{\mathsf{Hash}}$
$\mathbf{Q}_R^{\mathsf{ro}} = \mathbf{Q}_{\overline{R}}^{\mathsf{ro}}$; $\mathbf{Q}_R^{\mathsf{Hash}} = Q_{\overline{\mathsf{M}},q}$ *where* $q = \mathbf{Q}_{\overline{R}}^{\mathsf{Hash}}$; $\mathbf{Q}_{\overline{S}}^{\mathsf{ro}} = \mathbf{Q}_S^{\mathsf{ro}}$
$\mathbf{Q}_{\overline{S}}^{\mathsf{Hash}}$ *is the number of oracle queries of* M *in the execution of* $\mathrm{UCE}_{\mathsf{H}}^{S,D}$
$\mathbf{T}(\mathrm{Indiff}_{\mathsf{M},\overline{\mathsf{M}}}^{A}) = \mathbf{T}(\mathrm{UCE}_{\mathsf{H}}^{S,D})$; $\mathbf{T}(\mathrm{UCE}_{\overline{\mathsf{H}}}^{\overline{S},D}) = \mathbf{T}(\mathrm{UCE}_{\mathsf{H}}^{S,D})$
$\mathbf{T}(\mathrm{Reset}_S^R) = \mathbf{T}(\mathrm{Reset}_{\overline{S}}^{\overline{R}}) + T_{\overline{\mathsf{M}},q}$ *where* $q = \mathbf{Q}_{\overline{R}}^{\mathsf{Hash}}$
$\mathbf{T}(\mathrm{Indiff}_{\mathsf{M},\overline{\mathsf{M}}}^{B}) = \mathbf{T}(\mathrm{Reset}_S^R) + \mathbf{T}(\mathrm{Reset}_{\overline{S}}^{\overline{R}})$ □

We emphasize that Keyed-Indiff works in both the standard and the random oracle models. In particular if FIL family $\overline{\mathsf{H}}$ is $\mathsf{UCE}[\mathcal{S}^{\mathrm{xxx}}]$-secure in the standard model, then so is $\mathsf{Keyed\text{-}Indiff}[\mathsf{M},\overline{\mathsf{H}}]$, for $\mathrm{xxx} \in \{\mathrm{crs},\mathrm{srs}\}$. This resolves an open problem from [6] to construct UCE domain extenders in the standard model.

Instantiation. To obtain a concrete result that can be used in applications, we now instantiate $\overline{\mathsf{H}}$ above in a simple way, namely (1) $\overline{\mathsf{H}}.\mathsf{Kg}(1^\lambda)$ returns $hk \leftarrow\!\!\$\ \{0,1\}^\lambda$, and (2) $\overline{\mathsf{H}}.\mathsf{Ev}^{\mathsf{ro}}(1^\lambda, hk, x, 1^{\overline{\mathsf{H}}.\mathsf{ol}(\lambda)})$ returns $\mathsf{ro}(hk \,\|\, x)$. This is shown by BHK [6] to be UCE secure in the FIL-ROM for all forms of UCE they define. From Theorem 1 we obtain the following.

Theorem 2. *Let* $\overline{\mathsf{H}}$ *be constructed as above. Let* M *be a PRO such that* $\mathsf{M}.\mathsf{pil} = \overline{\mathsf{H}}.\mathsf{il}$ *and* $\mathsf{M}.\mathsf{pol} = \overline{\mathsf{H}}.\mathsf{ol}$. *Let* $\mathsf{H} = \mathsf{Keyed\text{-}Indiff}[\mathsf{M},\overline{\mathsf{H}}]$. *Let* $\mathrm{xxx} \in \{\mathrm{crs},\mathrm{srs}\}$.

Asymptotic result: H *is* $\mathsf{UCE}[\mathcal{S}^{\mathrm{crs}}]$*-secure.*

Concrete result: Let $\overline{\mathsf{M}}$ *be a simulator for* M. *Let* S *be an* N*-key source and* D *a distinguisher. We can construct a reset adversary* R *and an indifferentiability adversary* A *such that*

$$\mathsf{Adv}^{\mathsf{uce}}_{\mathsf{H},S,D}(\lambda) \le \mathsf{Adv}^{\mathsf{reset}}_{S,R}(\lambda) + 4N(\lambda)\cdot \mathsf{Adv}^{\mathsf{indiff}}_{\mathsf{M},\overline{\mathsf{M}},A}(\lambda) + \frac{2N(\lambda)\cdot q(\lambda) + N^2(\lambda)}{2^\lambda}$$

for every $\lambda \in \mathbb{N}$. *Furthermore,*

$\mathbf{Q}_A^{\mathsf{Prim}} = \mathbf{Q}_S^{\mathsf{Hash}}$; $\mathbf{Q}_A^{\mathsf{Func}} = \mathbf{Q}_R^{\mathsf{ro}} = \mathbf{Q}_D^{\mathsf{ro}}$; *and* $\mathbf{Q}_R^{\mathsf{Hash}} = Q_{\overline{\mathsf{M}},q}$, *where* $q = \mathbf{Q}_D^{\mathsf{ro}}$
$\mathbf{T}(\mathrm{Indiff}_{\mathsf{M},\overline{\mathsf{M}}}^{A}) = \mathbf{T}(\mathrm{Reset}_S^R) = \mathbf{T}(\mathrm{UCE}_{\mathsf{H}}^{S,D}) + T_{\overline{\mathsf{M}},q}$, *where* $q = \mathbf{Q}_D^{\mathsf{ro}}$ □

Theorem 2 is the one that can be used for the applications, namely to obtain FIL-ROM constructions for (possibly multi-stage) primitives that have been constructed using a VIL UCE function, such as those in BHK [6]. We simply instantiate the VIL UCE function with H given by Theorem 2. The broader paradigm to move from the VIL-ROM to the FIL-ROM is thus the following. Take a primitive with a VIL-ROM proof, and show that the random oracle can be UCE-instantiated. Then apply Theorem 2.

5 UCE from universal hashing

In this section, we show how almost universal hash functions can be used to build a domain extender for UCE.

$\underline{\mathsf{H.Kg}(1^\lambda)}$	$\underline{\mathsf{H.Ev^{ro}}(1^\lambda, hk, x, 1^\ell)}$
$fk \xleftarrow{\$} \mathsf{F.Kg}(1^\lambda)$; $\overline{hk} \xleftarrow{\$} \overline{\mathsf{H}}.\mathsf{Kg}(\lambda)$	$(\overline{hk}, fk) \leftarrow hk$; $u \leftarrow \mathsf{F.Ev}(1^\lambda, fk, x, 1^{\mathsf{F.ol}(\lambda)})$
$hk \leftarrow (\overline{hk}, fk)$; Return hk	$y \leftarrow \overline{\mathsf{H}}.\mathsf{Ev^{ro}}(1^\lambda, \overline{hk}, u, 1^\ell)$; Return y

Fig. 4. The $\mathsf{H} = \mathsf{AU\text{-}then\text{-}Hash}[\mathsf{F}, \overline{\mathsf{H}}]$ construction, built from a AU hash F and a FIL UCE-secure hash $\overline{\mathsf{H}}$.

AU HASH FAMILIES. For any function family F let

$$\mathbf{Coll1}_{\mathsf{F}}(\lambda, m) = \max_{|y|=\mathsf{F.ol}(\lambda), |x|\le m} \Big\{ \Pr_{fk \xleftarrow{\$} \mathsf{F.Kg}(1^\lambda)} [y = \mathsf{F.Ev}(1^\lambda, fk, x, 1^{\mathsf{F.ol}(\lambda)})] \Big\},$$

and define $\mathbf{Coll2}_{\mathsf{F}}(\lambda, m_0, m_1)$ as

$$\max \Big\{ \Pr_{fk \xleftarrow{\$} \mathsf{F.Kg}(1^\lambda)} [\mathsf{F.Ev}(1^\lambda, fk, x_0, 1^{\mathsf{F.ol}(\lambda)}) = \mathsf{F.Ev}(1^\lambda, fk, x_1, 1^{\mathsf{F.ol}(\lambda)})] \Big\};$$

the maximum is taken over distinct strings x_0, x_1 such that each $|x_i| \le m_i$. Let

$$\mathbf{Coll}_{\mathsf{F}}(\lambda, m_0, m_1) = \max\big\{ \mathbf{Coll2}_{\mathsf{F}}(\lambda, m_0, m_1), \mathbf{Coll1}_{\mathsf{F}}(\lambda, \min\{m_0, m_1\}) \big\} \ .$$

A hash family F is *almost universal* (AU) if $f(\lambda) = \mathbf{Coll}_{\mathsf{F}}(\lambda, m_0, m_1)$ is negligible for all polynomials m_0, m_1. This generalizes the Carter-Wegman notion of universal hashing [13].

A similar definition is given in [11], which is very useful when one needs to work with arbitrarily large input and short hash keys. In Section 6, we'll show how to concretely instantiate a very fast AU hash for $\lambda = 128$, from reduced-round AES and a classic polynomial-based universal hash. Define

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{F}}(\lambda, p, \sigma) = \max_{\ell \le p, \ell' \le p, m_1 + \cdots + m_\ell \le \sigma, m'_1 + \cdots + m'_{\ell'} \le \sigma} \Big\{ \sum_{i=1}^{\ell} \sum_{j=1}^{\ell'} \mathbf{Coll}_{\mathsf{F}}(\lambda, m_i, m'_j) \Big\} \ .$$

If F is AU then $\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{F}}(\lambda, p, \sigma)$ is negligible for all polynomials p and σ: since $\mathbf{Coll}(\lambda, \cdot, \cdot)$ is increasing in both arguments, it follows that $\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{F}}(\lambda, p, \sigma) \le p^2 \mathbf{Coll}_{\mathsf{F}}(\lambda, \sigma, \sigma)$.

UCE EXTENDER FROM AN AU HASH. We now describe a UCE extender from AU hash. Intuitively, one first uses the AU hash to condense the input, and then applies the resulting string to the (keyed) compression function. Formally, let $\overline{\mathsf{H}}$ be a hash function family of fixed input length, and F be a universal hash function family with $\mathsf{F.ol} = \overline{\mathsf{H}}.\mathsf{il}$ and $\mathsf{F.IL} = \mathbb{N}$. Consider the hash function family $\mathsf{H} = \mathsf{AU\text{-}then\text{-}Hash}[\mathsf{F}, \overline{\mathsf{H}}]$ as given in Fig. 4, with $\mathsf{H.OL} = \overline{\mathsf{H}}.\mathsf{OL}$ and $\mathsf{H.IL} = \mathbb{N}$. The construction essentially follows the widely used Carter-Wegman paradigm [24] Below, we show that $\mathsf{AU\text{-}then\text{-}Hash}[\mathsf{F}, \cdot]$ is also a domain extender for $\mathsf{UCE}[\mathcal{S}^{\mathrm{sup}}]$ security.

Theorem 3. *Let $\overline{\mathsf{H}}$ be a function family of fixed input length, and F be an AU hash function family with $\mathsf{F.ol} = \overline{\mathsf{H}}.\mathsf{il}$ and $\mathsf{F.IL} = \mathbb{N}$. Let $\mathsf{H} = \mathsf{AU\text{-}then\text{-}Hash}[\mathsf{F}, \overline{\mathsf{H}}]$.*

Asymptotic result: If $\overline{\mathsf{H}}$ *is* $\mathsf{UCE}[\mathcal{S}^{\mathrm{sup}}]$*-secure then so is* H.

Concrete result: Let S *be a* N*-key source,* D *a distinguisher, and* $\overline{P}$ *a predictor. We can construct a source* $\overline{S}$*, a distinguisher* $\overline{D}$*, and a predictor* P *such that*

$$\mathsf{Adv}^{\mathsf{uce}}_{\mathsf{H},S,D}(\lambda) \le \mathsf{Adv}^{\mathsf{uce}}_{\overline{\mathsf{H}},\overline{S},\overline{D}}(\lambda) + \mathsf{Adv}^{\mathsf{coll}}_{\mathsf{F}}(\lambda,p,\sigma) \tag{4}$$

$$\mathsf{Adv}^{\mathsf{pred}}_{\overline{S},\overline{P}}(\lambda) \le \sqrt{2q\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{F}}(\lambda,p,\sigma)} + \sqrt{q\mathsf{Adv}^{\mathsf{pred}}_{S,P}(\lambda)} \tag{5}$$

where $p = \mathbf{Q}^{\mathsf{Hash}}_{S}$*,* q *is the maximum of the size of* $\overline{P}$*'s output in the execution of* $\mathrm{Pred}^{\overline{P}}_{\overline{S}}$*, and* σ *is the maximum of the total length of* Hash *queries that* S *makes in* $\mathrm{UCE}^{S,D}_{\mathsf{H}}$*. Furthermore,*

$\mathbf{Q}^{\mathsf{ro}}_{\overline{S}} = \mathbf{Q}^{\mathsf{ro}}_{S}$; $\mathbf{Q}^{\mathsf{Hash}}_{\overline{S}} = \mathbf{Q}^{\mathsf{Hash}}_{S}$; $\mathbf{Q}^{\mathsf{ro}}_{\overline{D}} = \mathbf{Q}^{\mathsf{ro}}_{D}$
$\mathbf{T}(\mathrm{UCE}^{\overline{S},\overline{D}}_{\overline{\mathsf{H}}}) = \mathbf{T}(\mathrm{UCE}^{S,D}_{\mathsf{H}})$*, and* P *outputs a set of size at most* $\mathbf{Q}^{\mathsf{Hash}}_{S}$ □

We emphasize that AU-then-Hash works in both the standard and the random-oracle models. In particular If FIL family $\overline{\mathsf{H}}$ is $\mathsf{UCE}[\mathcal{S}^{\mathrm{sup}}]$-secure in the standard model then so is $\mathsf{AU\text{-}then\text{-}Hash}[\mathsf{F},\overline{\mathsf{H}}]$.

The intended applications for the $\mathsf{AU\text{-}then\text{-}Hash}[\mathsf{F},\cdot]$ transform, as listed in Fig. 1, use only a single hash key, that is, they only need $\mathsf{UCE}[\mathcal{S}^{\mathrm{sup}}\cap\mathcal{S}^{\mathrm{one}}]$ security, where $\mathcal{S}^{\mathrm{one}}$ is the class of 1-key sources. $\mathsf{AU\text{-}then\text{-}Hash}[\mathsf{F},\cdot]$ is also a domain extender for $\mathsf{UCE}[\mathcal{S}^{\mathrm{sup}} \cap \mathcal{S}^{\mathrm{one}}]$ security because the value of N is preserved.

Instantiation. So far we have assumed the existence of a fixed-input-length UCE-secure hash $\overline{\mathsf{H}}$. In the full version, we'll construct hash family $\mathsf{H}_{\mathsf{rom}}$, of variable output length, in the ROM, by using a pseudorandom permutation (PRP) E, which will be instantiated by AES. We conclude the following.

Theorem 4. *Let* F *be an AU hash function family with* $\mathsf{F.ol} = \mathsf{H_{rom}.il}$ *and* $\mathsf{F.IL} = \mathbb{N}$*. Let* $\mathsf{H} = \mathsf{AU\text{-}then\text{-}Hash}[\mathsf{F},\mathsf{H}_{\mathsf{rom}}]$.

Asymptotic result: H *is* $\mathsf{UCE}[\mathcal{S}^{\mathrm{sup}}]$*-secure.*

Concrete result: Let S *be an* N*-key source and* D *a distinguisher. We can construct a predictor* P *and a PRP adversary* A *such that*

$$\mathsf{Adv}^{\mathsf{uce}}_{\mathsf{H},S,D}(\lambda) \le 2\sqrt{q(\lambda)\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{F}}(\lambda,p(\lambda),\sigma(\lambda))} + \sqrt{q(\lambda)\mathsf{Adv}^{\mathsf{pred}}_{S,P}(\lambda)} + 2p(\lambda)\cdot\mathsf{Adv}^{\mathsf{prp}}_{\mathsf{E},A}(\lambda) + \frac{2s^2(\lambda)+N^2(\lambda)+q^2(\lambda)}{2^{\lambda}}$$

for every $\lambda \in \mathbb{N}$*, where* $p = \mathbf{Q}^{\mathsf{Hash}}_{S}$*;* $q = \mathbf{Q}^{\mathsf{ro}}_{S} + \mathbf{Q}^{\mathsf{ro}}_{D}$*;* σ *and* s *are the maximum of the total length of the first components and the total number of* λ*-bit blocks in the second components, respectively, of* Hash *queries in the execution of* $\mathrm{UCE}^{S,D}_{\mathsf{H}}$*. Furthermore*

$\mathbf{Q}^{\mathsf{LR}}_{A}$ *is maximum of the number of* λ*-bit blocks in the second component of a* Hash *query in* $\mathrm{UCE}^{S,D}_{\mathsf{H}}$
$\mathbf{T}(\mathrm{PRP}^{A}_{\mathsf{E}}) = \mathbf{T}(\mathrm{UCE}^{S,D}_{\mathsf{H}})$*, and* P *outputs a set of size at most* $\mathbf{Q}^{\mathsf{Hash}}_{S}$ □

6 Fast, parallelizable AU hash from reduced-round AES

We now show how to construct a fast parallelizable AU hash, which we call $\mathsf{F}_{\mathsf{aes4}}$. In this section, let $n = 128$, $C = 2^{15}$, and let r be a small integer, say $r = 5$. All function families in this section are concrete; the security parameter λ is hidden in the formulas, but implicitly, it is $\lambda = 128$. For any integer m, let $\|m\|_n$ denote $\lfloor m/n \rfloor + 1$. We'll first describe two building blocks: $\mathsf{F}_{\mathsf{poly}}$, a polynomial-based AU hash that operates on $\{0,1\}^*$, and $\mathsf{F}_{\mathsf{tree}}$, a highly efficient AU hash based on reduced-round AES that operates on $\{x \in (\{0,1\}^n)^+ : |x| \leq 2^r n\}$. We then show how to combine them to produce a highly efficient AU hash $\mathsf{F}_{\mathsf{aes4}}$ whose domain is $\{0,1\}^*$.

THE $\mathsf{F}_{\mathsf{poly}}$ CONSTRUCTION. We now describe a variant of a classic polynomial-based universal hash [13], which we call $\mathsf{F}_{\mathsf{poly}}$. Let $\mathsf{F}_{\mathsf{poly}}.\mathsf{ol} = n$. As described in the pseudocode below, the key fk is picked as a random element of $\mathrm{GF}(2^n)$. To hash, we parse the input string $x \in \{0,1\}^*$ to a unique sequence $(w_0, \ldots, w_m)$, where each $w_i \in \mathrm{GF}(2^n)$ and w_m is not the zero element. This is performed by (i) parse $v_0 \,\|\, \cdots \,\|\, v_m \leftarrow x \,\|\, 10^s 1$, where $s \in \mathbb{N}$ is the smallest number such that $s + |x| \equiv -2 \pmod{n}$ and each $|w_i| = n$, and (ii) let each w_i be the encoding of v_i in $\mathrm{GF}(2^n)$. Then, the hash is computed as $\sum_{i=0}^{m} w_i \cdot fk^i$.

$\underline{\mathsf{F}_{\mathsf{poly}}.\mathsf{Kg}()}$	$\underline{\mathsf{F}_{\mathsf{poly}}.\mathsf{Ev}(fk, x, 1^n)}$
$fk \leftarrow\!\!\$\ \mathrm{GF}(2^n)$	$(w_0, \ldots, w_m) \leftarrow x$; $y \leftarrow w_0$
Return fk	For $i = 1$ to m do $y \leftarrow y + w_i \cdot fk^i$
	Return y

Proposition 5. *(a) For any $m \in \mathbb{N}$, we have $\mathbf{Coll1}_{\mathsf{F}_{\mathsf{poly}}}(m) \leq \|m\|_n / 2^n$, and (b) for any $m_0, m_1 \in \mathbb{N}$, we have $\mathbf{Coll2}_{\mathsf{F}_{\mathsf{poly}}}(m_0, m_1) \leq \max\{\|m_0\|_n, \|m_1\|_n\}/2^n$.*

THE $\mathsf{F}_{\mathsf{tree}}$ CONSTRUCTION. Let $E : \{0,1\}^{4n} \times \{0,1\}^n \to \{0,1\}^n$ denote a function based on 4-round AES which works as follows. Parse the key K as the concatenation of n-bit substrings S_0, S_1, S_2, S_3, and let $S_4 = 0^n$. The input is initially xored with S_0, and each S_i is used as the subkey of the i-th AES round, for $i \in \{1,2,3,4\}$. One can build from E a hash of domain $\{n, 2n, 3n, \ldots, 2^r n\}$ as follows. Let Halve denote the following operation. On input $(K, x) \in \{0,1\}^{4n} \times (\{0,1\}^n)^*$, we partition x into n-bit blocks $x_1 \cdots x_m$. For every two consecutive blocks x_{2i-1} and x_{2i}, we compute $y_i \leftarrow E_K(x_{2i-1}) \oplus x_{2i}$. If m is odd then let $y_{\lceil m/2 \rceil} \leftarrow x_m$. Finally output $y_1 \,\|\, \cdots \,\|\, y_{\lceil m/2 \rceil}$. Consider the following tree-hash construction $\mathsf{F}_{\mathsf{tree}}$, with $\mathsf{F}_{\mathsf{tree}}.\mathsf{IL} = \{n, 2n, 3n, \ldots, 2^r n\}$ and $\mathsf{F}_{\mathsf{tree}}.\mathsf{ol} = n$.

$\underline{\mathsf{F}_{\mathsf{tree}}.\mathsf{Kg}()}$	$\underline{\mathsf{F}_{\mathsf{tree}}.\mathsf{Ev}(fk, x, 1^n)}$
For $i = 1$ to r do $K_i \leftarrow\!\!\$\ \{0,1\}^{4n}$	$z_0 \leftarrow x$; $(K_1, \ldots, K_r) \leftarrow fk$
$hk \leftarrow (K_1, \ldots, K_r)$; Return fk	For $i = 1$ to r do $z_i \leftarrow \mathsf{Halve}(K_i, z_{i-1})$
	Return z_r

Minematsu and Tsunoo [20] show that

$$\mathbf{Coll2}_{\mathsf{F}_{\mathsf{tree}}}(m_0, m_1) \leq \frac{Cr}{2^n} \tag{6}$$

for any $m_0, m_1 \leq 2^r$. We stress that the result in [20] makes no assumption on AES. This is based on the fact that four-round AES, with the subkeys chosen uniformly and independently, is an almost-xor-universal hash [18].

COMBINING $\mathsf{F}_{\mathsf{tree}}$ AND $\mathsf{F}_{\mathsf{poly}}$. One can "cascade" $\mathsf{F}_{\mathsf{tree}}$ and $\mathsf{F}_{\mathsf{poly}}$ to produce a hash $\mathsf{F}_{\mathsf{fast}}$ of domain $\{0,1\}^*$ as follows.

$\underline{\mathsf{F}_{\mathsf{fast}}.\mathsf{Kg}()}$	$\underline{\mathsf{F}_{\mathsf{fast}}.\mathsf{Ev}(fk, x, 1^n)}$	$\underline{\mathsf{Shrink}(fk_1, x)}$
$fk_1 \xleftarrow{\$} \mathsf{F}_{\mathsf{tree}}.\mathsf{Kg}()$	$(fk_1, fk_2) \leftarrow fk$	$w_1 w_2 \cdots w_k \leftarrow x$; $u_k \leftarrow w_k$
$fk_2 \xleftarrow{\$} \mathsf{F}_{\mathsf{poly}}.\mathsf{Kg}()$	$y \leftarrow \mathsf{Shrink}(fk_1, x)$	For $i = 1$ to $k-1$ do
Return (fk_1, fk_2)	$z \leftarrow \mathsf{F}_{\mathsf{poly}}.\mathsf{Ev}(fk_2, y, 1^n)$	$\quad u_i \leftarrow \mathsf{F}_{\mathsf{tree}}.\mathsf{Ev}(fk_1, w_i, 1^n)$
	Return z	$y \leftarrow u_1 \,\|\, \cdots \,\|\, u_k$; Return y

In the procedure Shrink above, we parse a string x as the concatenation of substrings $w_1, \ldots, w_k$, where the length of each w_i, with $i \leq k-2$, is exactly $2^r n$, and $|w_{k-1}| > 0$ is a multiple of n but does not exceed $2^r n$, and $0 \leq |w_k| < n-1$. Note that on a large input x, the hash F will make at most $(1-2^{-r})\lceil x/n \rceil$ calls on E, and then run $\mathsf{F}_{\mathsf{poly}}$ on a string of length about $|x|/2^r$.

Proposition 6. *For any $m_0, m_1 \in \mathbb{N}$, we have*

$$\mathbf{Coll}_{\mathsf{F}_{\mathsf{fast}}}(m_0, m_1) \leq \frac{Cr + \max\{\|m_0\|_n, \|m_1\|_n\}}{2^n}$$

USING WITH $\mathsf{AU\text{-}then\text{-}Hash}$. The hash $\mathsf{F}_{\mathsf{fast}}$ can't be used directly with the $\mathsf{AU\text{-}then\text{-}Hash}$ transform in Section 5, because the term $(q\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{F}_{\mathsf{fast}}}(p,\sigma))^{1/2}$ in Theorem 3 is about $(\sqrt{qp\sigma} + Crp\sqrt{q})/2^{n/2}$, which is inferior. The reason for this is that the output length of this hash is only n bits, which is too short. We therefore need to "double" the output length. Formally, given a hash family $\overline{\mathsf{F}}$, the family $\mathsf{F} = \mathsf{Double}[\overline{\mathsf{F}}]$, with $\mathsf{F.IL} = \overline{\mathsf{F}}.\mathsf{IL}$ and $\mathsf{F.ol} = 2\overline{\mathsf{F}}.\mathsf{ol}$, is constructed as follows.

$\underline{\mathsf{F}.\mathsf{Kg}()}$	$\underline{\mathsf{F}.\mathsf{Ev}(fk, x, 1^{\mathsf{F.ol}})}$
$fk_1, fk_2 \xleftarrow{\$} \overline{\mathsf{F}}.\mathsf{Kg}()$	$(fk_1, fk_2) \leftarrow fk$
$fk \leftarrow (fk_1, fk_2)$; Return fk	For $i = 1$ to 2 do $y_i \leftarrow \overline{\mathsf{F}}.\mathsf{Ev}(fk_i, x, 1^{\overline{\mathsf{F}}.\mathsf{ol}})$
	Return $y_1 \,\|\, y_2$

Let $\mathsf{F}_{\mathsf{aes4}}$ denote $\mathsf{Double}[\mathsf{F}_{\mathsf{fast}}]$. In Proposition 7 below, the term $(q\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{F}_{\mathsf{fast}}}(p,\sigma))^{1/2}$ in Theorem 3 is bounded by $(Crp\sqrt{2q} + 2(\|\sigma\|_n + p)\sqrt{pq})/2^n$, which is good.

Proposition 7. *For any p and σ, we have* $\mathsf{Adv}^{\mathsf{coll}}_{\mathsf{F}_{\mathsf{aes4}}}(p,\sigma) \leq \frac{2C^2r^2p^2 + 4p(\|\sigma\|_n + p)^2}{2^{2n}}$.

KEY LENGTH. The key material of $\mathsf{FastHash} = \mathsf{AU\text{-}then\text{-}Hash}[\mathsf{F}_{\mathsf{aes4}}, \mathsf{H}_{\mathsf{rom}}]$ is relatively large: 672B for $r = 5$. It's slightly bigger than that of some widely used schemes such as RSA [22] (256B). This is acceptable because the key is used as a public parameter.

Hash function	Setting	Speed (cycles per byte)		
		1MB	16MB	128MB
SHA-256 [1]		11.5	12.0	12.0
FastHash	sequential	2.1	2.2	2.2
	parallel - 12 threads	0.4	0.4	0.5

Fig. 5. Running time of the hash constructions. The first column lists the hash names, the second column lists the setting, namely sequential or parallel, along with the number of threads, and the last three columns list the running time on messages of sizes 1MB, 16MB, and 128MB respectively.

7 Implementation

In this section, we'll describe how to instantiate the AU hash $\mathsf{F}_{\mathsf{aes4}}$ in Section 6, and the FIL UCE-secure hash $\mathsf{H}_{\mathsf{rom}}$ in Section 5. We then compare the speed of FastHash, the resulting instantiation of AU-then-Hash$[\mathsf{F}_{\mathsf{aes4}}, \mathsf{H}_{\mathsf{rom}}]$, with a standard hash function, SHA-256. We first describe our choices for components and parameters to instantiate the construction, and then provide an overview of the implementation, before outlining the testing environment and test specifications. We also compare the convergent encryption (CE) MLE scheme [1] from FastHash and SHA-256. Our results indicate a speedup of 5.3x for our hash function over SHA-256 and 6.3x for CE in the sequential setting, and 24x and 20x speedups, respectively, once parallelism is enabled.

Instantiations. To instantiate $\mathsf{F}_{\mathsf{aes4}}$, we use the standard irreducible polynomial $p(x) = x^{127} + x^7 + x^2 + x + 1$ for multiplication over GF(2^{128}). For $\mathsf{H}_{\mathsf{rom}}$, the FIL RO is instantiated by the compression function of SHA-256, and the PRP by AES128.

Implementation. We implemented FastHash in C with inline assembly. We used Intel's library for multiplication over $\mathbb{GF}(2^{128})$ [3], Intel's optimized SHA256 implementation [1], and Intel's AES-NI library [2] for the code involving AES operations. We used the pthreads library for implementing threads for parallelization.

Setup. We performed experiments on an Intel Core i7-970 processor clocking at 3201 MHz with a 12288 KB L1 cache. The machine provides hardware support for SSE4 vector instructions, AES operations (AES-NI), and multiplication in $\mathbb{GF}(2^{128})$. Tests were compiled with gcc version 4.6 optimization level -O3, with support for SSE4 via -msse4 flag, AES-NI instructions through the -maes flag, $\mathbb{GF}(2^{128})$ multiplications via the -mpcmulqdq flag, and parallelization via the -pthread flag. We ran the tests in isolation, after turning off processor frequency scaling. We used the rdtsc instruction to count cycles.

[1] In CE [8], one first hashes the message x to derive a key K, and then runs AES-CTR on key K to encrypt x. To use FastHash on CE, one needs to use the CE variant of [6], in which AES-CTR on message m is replaced by $\mathsf{FastHash}(hk, K, 1^{|x|}) \oplus x$. Note that this doesn't give us any speed advantage over the standard version of CE, as the masking via FastHash is essentially AES-CTR. The only thing we gain is the abstraction of AES as part of the hash, so that one can apply $\mathsf{UCE}[\mathcal{S}^{\mathrm{sup}}]$.

MLE Scheme	Setting	Speed (cycles per byte)		
		1MB	16MB	128MB
CE implementation in [8]		22.1	22.3	22.6
CE[FastHash]	sequential	3.5	3.6	3.7
	parallel - 12 threads	1.2	1.1	1.1

Fig. 6. Running time of CE instantiations. The first column lists the instantiations, the second column lists the setting, namely sequential or parallel, along with the number of threads, and the last three columns list the running time (key generation + encryption) on messages of size 1MB, 16MB, and 128MB respectively.

Experiments. We measured the performance of instantiations of the hash functions (i.e. FastHash and SHA-256) as well as CE schemes based on these hash functions on messages of lengths 1MB, 16MB and 128MB. In each case, we measured the median running times of the different hash functions over 100 iterations, repeated this process 100 times and obtained the mean of the medians.

In the case of parallelizable constructions, viz. FastHash and CE[FastHash], we ran tests with multiple levels of parallelism, starting from single-threaded, serial constructions, and increasing the number of threads until we reached a point of thrashing where the performance starts to deteriorate because of other bottlenecks in the system. We report both the single-thread sequential running time, and the optimal parallel running time along with the optimal number of threads. In the latter case, the reported time does not include the time to create and destroy the threads.

In Fig. 5, we report the median running times of the hash function instantiations, in cycles per byte. We compare these times with the best times reported for SHA-256 on similar processors [1]. Our construction achieves substantially better running times. On messages of 1MB, SHA runs at 11.5 cycles per byte, but our instantiation runs more than 5.3 times faster, at a cost of 2.1 cycles per byte. With parallelism, we achieve much better speeds, below one cycle per byte.

In Fig. 6, we demonstrate the benefits of having faster hash functions by comparing the speeds of CE implemented with FastHash with the implementation of CE by SHA-256 and AES-CTR in [8]. Our experiments show that CE[FastHash], even in the sequential setting, is about 6.3x faster than the speeds reported in [8]. When parallelism enabled, we achieve about 20x speedup.

Acknowledgments

Work done while Keelveedhi was a PhD student at UCSD. The authors were supported in part by NSF grants CNS-1116800 and CNS-1228890.

References

1. Fast SHA-256 Implementations on Intel Architecture Processors. goo.gl/Hh81eB.

2. Intel AESNI Library. `goo.gl/l2czm1`.
3. Intel Carry-Less Multiplication Instruction and its Usage for Computing the GCM Mode. `goo.gl/qJLrF1`.
4. B. Barak, Y. Dodis, H. Krawczyk, O. Pereira, K. Pietrzak, F.-X. Standaert, and Y. Yu. Leftover hash lemma, revisited. In *CRYPTO 2011*, Springer, 2011.
5. M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO 2007*, Springer, 2007.
6. M. Bellare, V. T. Hoang, and S. Keelveedhi. Instantiating random oracles via UCEs. Cryptology ePrint Archive, Report 2013/424, 2013. Preliminary version appeared in *CRYPTO 2013*, Springer, 2013.
7. M. Bellare, V. T. Hoang, and S. Keelveedhi. Cryptography from compression functions: The UCE bridge to the ROM. Cryptology ePrint Archive, 2014.
8. M. Bellare, S. Keelveedhi, and T. Ristenpart. Message-locked encryption and secure deduplication. In *EUROCRYPT 2013*, Springer, 2013.
9. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93*, ACM, 1993.
10. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT 2006*, Springer, 2006.
11. J. Black and P. Rogaway. CBC MACs for arbitrary-length messages: The three-key constructions. *Journal of Cryptology*, 18(2):111–131, Apr. 2005.
12. C. Brzuska, P. Farshim, and A. Mittelbach. Indistinguishability obfuscation and uces: The case of computationally unpredictable sources. Cryptology ePrint Archive, Report 2014/099. To appear in *CRYPTO 2014*, Springer, 2014.
13. L. Carter and M. Wegman. Universal classes of hash functions. *Journal of computer and system sciences*, 18(2):143–154, 1979.
14. J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In *CRYPTO 2005*, Springer, Aug. 2005.
15. G. Demay, P. Gazi, M. Hirt, and U. Maurer. Resource-restricted indifferentiability. In *EUROCRYPT 2013*, Springer, 2013.
16. Y. Dodis, T. Ristenpart, and T. Shrimpton. Salvaging Merkle-Damgård for practical applications. In *EUROCRYPT 2009*, Springer, 2009.
17. J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.
18. L. Keliher and J. Sui. Exact maximum expected differential and linear probability for two-round advanced encryption standard. *IET Information Security*, 1(2):53–57, 2007.
19. U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *TCC 2004*, Springer, 2004.
20. K. Minematsu and Y. Tsunoo. Provably secure macs from differentially-uniform permutations and aes-based implementations. In *FSE 2006*, Springer, 2006.
21. A. Mittelbach. Salvaging indifferentiability in a multi-stage setting. In *EUROCRYPT 2014*, Springer, 2014.
22. PKCS #1: RSA cryptography standard. RSA Data Security, Inc., Sept. 1998. Version 2.0.
23. T. Ristenpart, H. Shacham, and T. Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In *EUROCRYPT 2011*, Springer, 2011.
24. M. N. Wegman and L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22:265–279, 1981.