Indistinguishability Obfuscation and UCEs: The Case of Computationally Unpredictable Sources

Christina Brzuska[1] Pooya Farshim[2] Arno Mittelbach[2]

[1]Tel Aviv University, Israel
[2]Darmstadt University of Technology, Germany

Abstract. Random oracles are powerful cryptographic objects. They facilitate the security proofs of an impressive number of practical cryptosystems ranging from KDM-secure and deterministic encryption to point-function obfuscation and many more. However, due to an uninstantiability result of Canetti, Goldreich, and Halevi (STOC 1998) random oracles have become somewhat controversial. Recently, Bellare, Hoang, and Keelveedhi (BHK; CRYPTO 2013 and ePrint 2013/424, August 2013) introduced a new abstraction called Universal Computational Extractors (UCEs), and showed that they suffice to securely replace random oracles in a number of prominent applications, including all those mentioned above, without suffering from the aforementioned uninstantiability result. This, however, leaves open the question of constructing UCEs in the standard model.

We show that the existence of indistinguishability obfuscation (iO) implies (non-black-box) attacks on all the definitions that BHK proposed within their UCE framework in the original version of their paper, in the sense that no concrete hash function can satisfy them. We also show that this limitation can be overcome, to some extent, by restraining the class of admissible adversaries via a *statistical* notion of unpredictability. Following our attack, BHK (ePrint 2013/424, September 2013), independently adopted this approach in their work.

In the updated version of their paper, BHK (ePrint 2013/424, September 2013) also introduce two other novel source classes, called *bounded parallel sources* and *split sources*, which aim at recovering the computational applications of UCEs that fall outside the statistical fix. These notions keep to a computational notion of unpredictability, but impose structural restrictions on the adversary so that our original iO attack no longer applies. We extend our attack to show that indistinguishability obfuscation is sufficient to also break the UCE security of any hash function against bounded parallel sources. Towards this goal, we use the *randomized encodings* paradigm of Applebaum, Ishai, and Kushilevitz (STOC 2004) to parallelize the obfuscated circuit used in our attack, so that it can be computed by a bounded parallel source whose second stage consists of constant-depth circuits. We conclude by discussing the composability and feasibility of hash functions secure against split sources.

Keywords. Randomized encodings, Obfuscation, UCE, Random oracle

1 Introduction

Since their formal introduction in the seminal paper of Bellare and Rogaway [BR93], random oracles have found extensive use across a wide spectrum of cryptographic protocols. Their versatility has lead researchers to seek for a unified formalization of their useful properties, hoping that such a definition could be eventually realized. Canetti, Goldreich, and Halevi [CGH98] proposed such a definition, but somewhat disappointingly, also proved a negative result which ruled out instantiations of random oracles in *arbitrary* (perhaps artificial) cryptographic protocols by *any* keyed hash functions. This negative result was subsequently extended in a number of works [Nie02, GK03, DOP05, KP09, BBP04, CGH04].

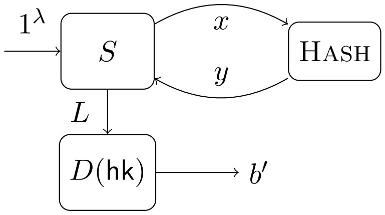

Figure 1: Schematic of the UCE game. Source S has access to Hash, which returns real or ideal hash values, and leaks L to a distinguisher D. The latter additionally gets the hash key and outputs a bit b'.

```
Main $\mathrm{UCE}_{\mathsf{H}}^{S,D}(\lambda)$
$b \leftarrow\$ \{0,1\}$; $\mathsf{hk} \leftarrow\$ \mathsf{H.Kg}(1^\lambda)$
$L \leftarrow\$ S^{\textsc{Hash}}(1^\lambda)$
$b' \leftarrow\$ D(1^\lambda, \mathsf{hk}, L)$
return $(b = b')$
```

```
Hash$(x)$
if $T[x] = \bot$ then
  if $b = 1$ then
    $T[x] \leftarrow \mathsf{H.Ev}(1^\lambda, \mathsf{hk}, x)$
  else $T[x] \leftarrow\$ \{0,1\}^{\mathsf{H.ol}(\lambda)}$
return $T[x]$
```

Figure 2: Pseudocode for the UCE game. Here $\mathsf{H.ol}(\lambda)$ is a function which specifies the length of hash values.

UCE security. Bellare, Hoang, and Keelvedhi (BHK) [BHK13a, BHK13b, BHK13c][1] revisited the above question and formulated an attractive new security notion called *Universal Computational Extractor* (UCE). They were able to apply their framework to an interesting and diverse set of security goals, which included among other things, security under key-dependent attacks, security under related-key attacks, simultaneous hardcore bits, point-function obfuscation, garbling schemes, proofs of storage, and deterministic encryption. Recently, Matsuda and Hanaoka [MH14] were able to use UCEs to also build CCA-secure public-key encryption schemes.

The UCE framework comes in two versions: a single-key version (UCE) and a multi-key version (mUCE). For a keyed hash function H, single-key UCE security is defined via a two-stage security game consisting of algorithms S and D, called the *source* and the *distinguisher*, respectively. In the first stage, the source is given access to an oracle Hash that, depending on a challenge bit b, implements either a random oracle or the concrete hash function with a randomly chosen key hk. The source terminates with some leakage L, which is then communicated together with hk to the distinguisher D. The distinguisher's goal is to guess the bit b, i.e., guess whether the source interacted with the random oracle or the hash function. The UCE advantage of the pair (S, D) is defined as the probability of returning the correct answer scaled away from one-half. (The stronger multi-key version is defined analogously by introducing Hash oracles for multiple keys and providing the keys together with leakage to the distinguisher.) We summarize this interaction schematically in Figure 1, and give the pseudocode in Figure 2. We refer the reader to the original work for an excellent philosophical perspective on this framework.

Without any restrictions UCE security cannot be achieved: the source can simply leak one of its oracle queries together with the corresponding answer to the distinguisher, which then can locally compute the hash value on the queried point (the distinguisher knows the hash key) and compare it to the leaked hash value. Thus, the source needs to be somehow restricted, and this restriction forms the actual UCE definition: for a source class $\mathcal{S}$ we denote the UCE assumption with sources restricted to $\mathcal{S}$ by UCE[$\mathcal{S}$]. Prior to our work, BHK proposed two source classes via *unpredictability* and *reset security* conditions, which in turn gave rise to two notions called UCE1 and UCE2, respectively.

The UCE1 notion [BHK13a, BHK13b] is defined using an unpredictability game which requires that when the source is run with a *random oracle*, its leakage does not *computationally* reveal any of its queries. This is formalized by requiring that the probability of any efficient predictor P in guessing a query of S when given L is negligible. Such a source is then called unpredictable, and leads to the following definition of UCE1 security: a hash function is UCE1 secure if the advantage of all efficient, unpredictable sources S, and all efficient distinguishers D in the UCE game is negligible. The stronger notion of UCE2 security is defined analogously by requiring that the source satisfies the weaker requirement of reset security.

[1] Citation [BHK13a] refers to the CRYPTO 2013 proceedings version, [BHK13b] refers to its full version on Cryptology ePrint Archive from August 2013 prior to communicating our basic iO attack (presented in this paper), and [BHK13c] refers to the updated version from September/October 2013.

Following our attack (that we describe next and that we communicated to BHK in August 2013 [BHK13d]), the UCE1 and UCE2 notions were revised in [BHK13c] and additional restrictions on sources were imposed. We will be discussing these shortly, after presenting our first attack.

An attack on UCE1 (and UCE2). Our first attack, described in Section 3, targets the original UCE notions UCE1 and UCE2, and is based on a recent breakthrough in the construction of obfuscation schemes. Garg et al. [GGH$^+$13] give a candidate construction for the so-called notion of indistinguishability obfuscation [BGI$^+$01] based on intractability assumptions related to multi-linear maps. Our attack shows that any UCE1 construction would need to falsify one of these assumptions. Put differently, if indistinguishability obfuscation exists, then UCE1 security (and hence also the stronger UCE2 security) cannot be achieved.

Roughly speaking, a secure indistinguishability obfuscation (iO) scheme assures that the obfuscations of any two circuits that implement the same function are computationally indistinguishable. Our attack uses this primitive as follows. The source picks a random point x, and queries it to Hash to get y. It then prepares an iO of the Boolean circuit $(\mathsf{H}(\cdot, x) = y)$, and leaks it to the distinguisher as L. The distinguisher now plugs the hash key hk into this obfuscated circuit and returns whatever the circuit outputs. It is easy to see that the distinguisher recovers the challenge bit correctly with an overwhelming probability. What is less clear, however, is whether or not the source is unpredictable. Recall that the unpredictability game operates with respect to a random oracle. Let us now assume, for simplicity, that $|\mathsf{hk}| < |y|/2$ (we will not need to rely on this assumption in our full attack). For any x, there are at most $2^{|\mathsf{hk}|}$ possible values for $\mathsf{H}(\mathsf{hk}, x)$, and a random y would be one of them with probability at most $2^{|\mathsf{hk}|}/2^{|y|} < 2^{-|y|/2}$, which is negligible. Consequently, the obfuscated circuit implements the constant *zero* function with overwhelming probability. This allows us to apply the security of the obfuscator to conclude the attack: the obfuscated circuit does not leak any more information about x than the zero function would, and since x was chosen randomly, it remains hidden from the view of any efficient predicator.

Salvaging UCE. Assuming the existence of indistinguishability obfuscation, we ask to what extent UCE can be salvaged. That is, do there exist other UCE assumptions that allow recovering (some of) the originally presented applications? We partially salvage UCEs by modifying the unpredictability condition and letting the predictor run in *unbounded* time. This statistical notion of unpredictability restricts the class of admissible sources such that the source implementing the iO attack falls outside it: an unbounded predictor can reverse engineer the *computationally* secure obfuscator. This modification is validated by the work of Goldwasser and Rothblum [GR07] who show that a statistical analogue of iO is impossible unless the polynomial hierarchy collapses to its second level. As we discuss in Section 3.2, a large number of interesting applications (such as KDM and RKA security) survives under this definition.

After communicating our attack, BHK independently suggested the statistical patch [BHK13c]. In the revised version of their paper [BHK13c], they recast their proofs of security to rely only on statistical unpredictability for all applications where this is possible. We refer to [BHK13c] for details on the applications that can be salvaged by statistical UCE1. As mentioned earlier, not all applications can be salvaged by statistical unpredictability. Hence, BHK also present two additional UCE notions based on computational unpredictability, which together with the statistical patch allowed them to fully recover their original set of applications in light of the aforementioned iO attack. We discuss these next.

Computational UCE. Some applications discussed in [BHK13a, BHK13b], specifically hardcore functions, deterministic public-key encryption (D-PKE), message-locked encryption (MLE), and OAEP rely on computational unpredictability in an intrinsic way; that is, the reduction only works if the predictor is bound to run in polynomial time. For instance, the source presented in [BHK13a, BHK13b] for D-PKEs produces leakage which contains encryptions of messages that have been sent to the Hash oracle. An unbounded predictor can easily decipher the ciphertexts and predict $\textsc{Hash}$ queries of the source.

Following the above attack, in the updated version of their paper, BHK [BHK13c] propose two novel UCE notions by imposing additional restrictions on the way the source operates, while keeping the original *computational* unpredictability game. The goal here is that these restrictions are sufficiently strong to circumvent our attack, but weak enough so that successful security reductions can be established.

To recover D-PKEs, MLEs, and OAEP, BHK propose a new UCE assumption based on computational unpredictability restricted to so-called *bounded parallel sources.* Such a source splits into two stages S_0 and S_1. In the first phase, algorithm S_0 prepares a vector of strings. In the second phase, independent instances of S_1 for each entry in the previously prepared vector are run in parallel. Each instance gets access to the $\textsc{Hash}$ oracle and their combined outputs make up the final leakage. To circumvent our attack two restrictions on S_1 are imposed: its runtime and number of $\textsc{Hash}$ queries (per instance). The idea here is that computing the obfuscation of a hash function is "too costly," and hence the attack cannot be mounted.

In Section 4 we show that this refined notion still falls prey to a similar, but somewhat more complex attack. The idea is to split the iO attack into two stages consisting of a high-complexity first stage and a parallelizable second stage. To this end, we use the powerful *randomized encodings* paradigm of Applebaum, Ishai, and Kushilevitz [AIK04] to bring down the complexity of the second stage of the attack. The randomized encoding $\hat{f}(x;r)$ of $f(x)$ is simply an encoding of $f(x)$ such that a decoder dec can retrieve the original value $f(x)$ from it, i.e., $\mathsf{dec}(\hat{f}(x;r)) = f(x)$. In addition, a randomized encoding specifies an efficient simulator Sim such that for all x the distributions $\hat{f}(x;r)$ over uniformly chosen r and $\mathsf{Sim}(f(x))$ are computationally indistinguishable. These properties combined allow us to show that we can adapt our original attack such that the source does not leak the obfuscated circuit but rather a randomized encoding of it. This alone, however, is still not enough for an attack with the restrictions of bounded parallel sources. Finally, we utilize a special form of *decomposable* randomized encodings [IKOS08] to realize an attack. Such encodings have the property that each output bit of $\hat{f}(x;r)$ depends on at most a *single* bit of x (but possibly on the entire string r). The randomized encoding of Applebaum, Ishai, and Kushilevitz [AIK06] is decomposable and supports all functions in $\mathcal{P}/poly$. We show how to use such an encoding scheme to split the computation of the encoding into two phases: a complex first preprocessing phase which does not depend on the actual input and a very simple second stage which can be parallelized and where each parallel instance essentially only has to drop one of two bits. We show that this second stage (which will correspond to S_1) can be implemented by constant-depth circuits consisting only of very few gates. This application of decomposable randomized encodings could be of interest also in other scenarios where efficiently computing an encoding is important and preprocessing is possible.

While bounded parallel sources are sufficient to also recover simultaneous hardcore functions, BHK propose a second, simpler UCE assumption based on *split sources.* A split source consists of two parts S_0 and S_1, in which each part independently contributes to the leakage sent to the distinguisher. The idea is that none of these sub-sources gets direct access to the $\textsc{Hash}$ oracle. Rather, algorithm S_0 defines the queries (without access to any hash values) and algorithm S_1 gets to see the hash values but not the queries. As for our attack, note that the associated source needs to know both the query x and its hash value $y \leftarrow \textsc{Hash}(x)$ in order to compute the circuit $(\mathsf{H}(\cdot, x) = y)$. In Section 5, we discuss split sources in a larger context and present necessary conditions for a hash function to achieve split-source UCE security. For example, we show that in order to prove the security of a hash function H, one needs to show that the function that maps x to the obfuscation of the circuit $\mathsf{H}(\cdot, x)$ must not be one way. We also discuss intricacies regarding composition of such functions with one-way permutations, and show that such a composition does not harm standard notions such as collision resistance, pseudorandomness (and indeed statistical UCE1 security) but provably fails for split-source security. We present this discussion in Section 5.

```
MAIN $\mathrm{UCE}_{\mathsf{H}}^{S,D}(\lambda)$
$b \leftarrow\!\!\$\ \{0,1\}$; $\mathsf{hk} \leftarrow\!\!\$\ \mathsf{H.Kg}(\lambda)$
$L \leftarrow\!\!\$\ S^{\textsc{Hash}}(1^\lambda)$
$b' \leftarrow\!\!\$\ D(1^\lambda, \mathsf{hk}, L)$
return $(b = b')$

HASH$(x)$
if $T[x] = \perp$ then
    if $b = 1$ then $T[x] \leftarrow \mathsf{H.Ev}(1^\lambda, \mathsf{hk}, x)$
    else $T[x] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{H.ol}(\lambda)}$
return $T[x]$
```

```
MAIN $\mathsf{Pred}_S^P(\lambda)$
$\mathsf{done} \leftarrow$ false; $Q \leftarrow \emptyset$
$L \leftarrow\!\!\$\ S^{\textsc{Hash}}(1^\lambda)$; $\mathsf{done} \leftarrow$ true
$Q' \leftarrow\!\!\$\ P^{\textsc{Hash}}(1^\lambda, L)$
return $(Q \cap Q' \neq \emptyset)$

HASH$(x)$
if $\mathsf{done}$ = false then $Q \leftarrow Q \cup \{x\}$
if $T[x] = \perp$ then
    $T[x] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{H.ol}(\lambda)}$
return $T[x]$
```

```
MAIN $\mathsf{Reset}_S^R(\lambda)$
$\mathsf{Dom} \leftarrow \emptyset$; $L \leftarrow\!\!\$\ S^{\textsc{Hash}}(1^\lambda)$; $b \leftarrow\!\!\$\ \{0,1\}$
if $b = 0$ then
    for $(x, \ell) \in \mathsf{Dom}$ do
        $T[x] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{H.ol}(\lambda)}$
$b' \leftarrow\!\!\$\ R^{\textsc{Hash}}(1^\lambda, L)$; return $(b' = b)$

HASH$(x)$
$\mathsf{Dom} \leftarrow \mathsf{Dom} \cup \{x\}$
if $T[x] = \perp$ then $T[x] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{H.ol}(\lambda)}$
return $T[x]$
```

Figure 3: The UCE security game together with the unpredictability and reset-security games.

To conclude, although UCEs strengthen our confidence in the security of many practical schemes in the random-oracle model, our attacks highlight the need for a thorough assessment of definitional choices that can be made within the UCE framework. This assessment, in addition to instantiability questions, should also include studying concrete instantiations of UCEs such as the SHA family [Nat12] in HMAC mode, as suggested by BHK [BHK13a].

2 Preliminaries

Notation. We denote by $\lambda \in \mathbb{N}$ the security parameter, which is implicitly given to all algorithms (if not explicitly stated so) in the unary representation 1^λ. By $\{0,1\}^\ell$ we denote the set of all bit-strings of length ℓ, and by $\{0,1\}^*$ the set of all bit-strings of finite length. For two strings $x_1, x_2 \in \{0,1\}^*$ their concatenation is written as $x_1 \| x_2$. The length of x is denoted by $|x|$ and $x[i]$ is the i-th bit of x. For a finite set X, we denote the action of sampling x uniformly at random from X by $x \leftarrow\!\!\$\ X$, and denote the cardinality of X by $|X|$. Algorithms are assumed to be randomized, unless otherwise stated. We call an algorithm efficient or PPT if it runs in time polynomial in the security parameter. By $y \leftarrow \mathcal{A}(x; r)$ we denote that y was output by algorithm $\mathcal{A}$ on input x and randomness r. If $\mathcal{A}$ is randomized and no randomness is specified, then we assume that $\mathcal{A}$ is run with freshly sampled uniform random coins, and write this is as $y \leftarrow\!\!\$\ \mathcal{A}(x)$. We often refer to algorithms, or tuples of algorithms, as adversaries. We say a function $\mathrm{negl}(\lambda)$ is negligible if $|\mathrm{negl}(\lambda)| \in \lambda^{-|\omega(1)|}$. In this paper we deploy the game-playing framework of Bellare and Rogaway [BR06] with the augmented game procedures described in [RSS11].

Syntax of hash functions. In line with [BHK13a], we consider the following formalization of hash functions. A function family H is a five tuple of PPT algorithms $(\mathsf{H.Kg}, \mathsf{H.Ev}, \mathsf{H.kl}, \mathsf{H.il}, \mathsf{H.ol})$ as follows. The algorithms $\mathsf{H.kl}$, $\mathsf{H.il}$, and $\mathsf{H.ol}$ are deterministic and on input 1^λ define the key length, input length, and output lengths, respectively. (We have adopted the simplified notion from [BHK13a] here.) The key generation algorithm $\mathsf{H.Kg}$ gets the security parameter 1^λ as input and outputs a key $\mathsf{hk} \in \{0,1\}^{\mathsf{H.kl}(\lambda)}$. The deterministic evaluation algorithm $\mathsf{H.Ev}$ takes as input the security parameter 1^λ, a key hk, a message $x \in \{0,1\}^{\mathsf{H.il}(\lambda)}$ and generates a hash value $\mathsf{H.Ev}(1^\lambda, \mathsf{hk}, x) \in \{0,1\}^{\mathsf{H.ol}(\lambda)}$.

UCE game. Let $\mathsf{H} = (\mathsf{H.Kg}, \mathsf{H.Ev}, \mathsf{H.kl}, \mathsf{H.il}, \mathsf{H.ol})$ be a hash function and (S, D) be a pair of PPT algorithms. We define the UCE advantage of (S, D) against H through

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

where game $\mathrm{UCE}_{\mathsf{H}}^{S,D}(\lambda)$ is shown in Figure 3 on the left.

Unpredictability. A source S is called *computationally unpredictable* if the advantage of any PPT predictor P defined by

$$\mathsf{Adv}^{\mathsf{pred}}_{S,P}(\lambda) := \Pr\left[\,\mathsf{Pred}^{P}_{S}(\lambda)\right]$$

is negligible, where game $\mathsf{Pred}^{P}_{S}(\lambda)$ is shown in Figure 3 in the middle. We denote the class of all computationally unpredictable sources by $\mathcal{S}^{\mathrm{cup}}$.

UCE security. We say a hash function H is UCE1 secure if for all computationally unpredictable PPT sources S and all PPT distinguishers D the advantage $\mathsf{Adv}^{\mathsf{uce}}_{\mathsf{H},S,D}(\lambda)$ is negligible. In the later version of their paper [BHK13c], BHK refer to UCE1 as UCE[$\mathcal{S}^{\mathrm{cup}}$]. BHK introduce a stronger version called UCE2 which is based on the reset-security game $\mathsf{Reset}^{R}_{S}(1^{\lambda})$ shown in Figure 3 on the right. We refer the reader to [BHK13b] for the details, but note here that UCE2 security implies UCE1 security and, thus, any attack on UCE1 also applies to UCE2.

We discuss the revised UCE assumptions introduced in [BHK13c], namely those for *bounded parallel sources* and *split sources*, in Sections 4 and 5, respectively.

Indistinguishability obfuscation. Roughly speaking, an indistinguishability obfuscation (iO) scheme ensures that the obfuscations of any two functionally equivalent circuits are computationally indistinguishable. Indistinguishability obfuscation was originally proposed by Barak et al. [BGI+01] as a potential weakening of virtual-black-box obfuscation. We recall the definition from [GGH+13]. A PPT algorithm iO is called an *indistinguishability obfuscator* for a circuit class $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ if the following conditions are satisfied:

- Correctness. For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, and for all inputs x we have that
$$\Pr\left[C'(x) = C(x) : C' \leftarrow_{\$} \mathsf{iO}(1^{\lambda}, C)\right] = 1 \ .$$
- Security. For any PPT distinguisher D, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\lambda$ such that $C_0(x) = C_1(x)$ on all inputs x the following distinguishing advantage is negligible:
$$\mathsf{Adv}^{\mathsf{io}}_{\mathsf{iO},D,C_0,C_1}(\lambda) := \Pr\left[D(\mathsf{iO}(1^{\lambda}, C_1)) = 1\right] - \Pr\left[D(\mathsf{iO}(1^{\lambda}, C_0)) = 1\right] \ .$$

With their recent candidate construction for indistinguishability obfuscation, Garg et al. [GGH+13] have revived interest in the study of obfuscation schemes (see, for example, [SW13, BR13, HSW14, BR14a, BR14b, GGHR14, BGK+14, BCP14] and the references therein). Garg et al. prove that under an intractability assumption related to multi-linear maps their construction yields an indistinguishability obfuscator for all circuits in $\mathcal{NC}^1$. Additionally, assuming a perfectly correct fully homomorphic encryption scheme and a perfectly sound non-interactive witness-indistinguishable proof system, they also show how their obfuscation scheme can be bootstrapped to support any polynomial-size circuit. In a recent work, Barak et al. [BGK+14] have further simplified the construction and showed that it is secure against all generic multi-linear attacks.

3 UCE1 and UCE2 Security

In this section we formalize our iO attack on the UCE1 (and hence the stronger UCE2) security of *any* concrete hash function. We also propose a fix to these notions which avoids the attack while still being applicable to a number of cryptosystems.

3.1 The iO attack

Our attack uses an indistinguishability obfuscation scheme in a black-box way, but is non-black-box as it relies on the code of the hash function for obfuscation. (Therefore, the attack does not contradict the positive feasibility of BHK in the random-oracle model.) We stress that the complexity of running our attack, although high, is polynomial, and will benefit from future advances in the construction of iO schemes.

Theorem 3.1 (UCE1 infeasibility)**.** *If indistinguishability obfuscation exists, then* UCE1 *security cannot be achieved in the standard model.*

Proof. Let H be a UCE1-secure hash function family. Let us assume for now that $\mathsf{H.ol}(\lambda) \geq 2 \cdot \mathsf{H.kl}(\lambda)$, that is, the output length of the hash function is at least twice the size of a hash key. (We will be dropping this condition shortly.) Define a source S which generates a random value $x \leftarrow_{\$} \{0,1\}^{\mathsf{H.il}(\lambda)}$ and computes $y \leftarrow \textsc{Hash}(x)$. It then constructs the Boolean circuit

$$C_{\lambda,\mathsf{H},x,y}(\cdot) := (\mathsf{H.Ev}(1^\lambda, \cdot, x) = y) \,.$$

The source S passes on an encoding of circuit $C_{\lambda,\mathsf{H},x,y}(\cdot)$ as leakage L to the distinguisher. We will later use obfuscation to ensure that x is not leaked by the encoding of $C_{\lambda,\mathsf{H},x,y}(\cdot)$ (this is needed for unpredictability). The distinguisher D recovers circuit $C_{\lambda,\mathsf{H},x,y}(\cdot)$ from the leakage L, and computes $b' \leftarrow C_{\lambda,\mathsf{H},x,y}(\mathsf{hk})$ using the given hash key hk, and returns b'. The UCE1 adversary (S, D) has advantage $1 - 2^{-\mathsf{H.ol}(\lambda)}$: When the source is run with oracle access to $\mathsf{H.Ev}(1^\lambda, \mathsf{hk}, \cdot)$, the circuit always returns 1. When S interacts with a random oracle, y coincides with $\mathsf{H.Ev}(1^\lambda, \mathsf{hk}, x)$ with probability $2^{-\mathsf{H.ol}(\lambda)}$.

Now let iO be an indistinguishability obfuscator. Instead of leaking circuit $C_{\lambda,\mathsf{H},x,y}(\cdot)$, we let S compute an obfuscation of the circuit and output $L \leftarrow_{\$} \mathsf{iO}(C_{\lambda,\mathsf{H},x,y}(\cdot))$. By the correctness property of the obfuscator, distinguisher D, as before, has an overwhelming advantage in guessing the challenge bit correctly. It remains to show that the adapted source S is unpredictable.

<span style="font-variant:small-caps">Unpredictability.</span> Let P be a predictor in the $\mathsf{Pred}_S^P(\lambda)$ game. We bound the success probability of P based on the security of the indistinguishability obfuscator iO via a sequence of three games as follows.

$\mathsf{Game}_0(\lambda)$**:** is identical to the computational unpredictability game $\mathsf{Pred}_S^P(\lambda)$ (see Figure 3).

$\mathsf{Game}_1(\lambda)$**:** is similar to the previous game except that the game terminates if there exists an $\mathsf{hk} \in \{0,1\}^{\mathsf{H.kl}(\lambda)}$ such that $C_{\lambda,\mathsf{H},x,y}(\mathsf{hk}) = 1$. We write $(x, y) \in \mathsf{Bad}(\lambda)$ if for values (x, y) such an hk exists for which $C_{\lambda,\mathsf{H},x,y}(\mathsf{hk}) = 1$.

$\mathsf{Game}_2(\lambda)$**:** is similar to the previous game except that S now leaks $\mathsf{iO}(Z_\lambda(\cdot))$, where $Z_\lambda(\cdot)$ denotes the constant zero circuit.

By the fundamental lemma of the game-playing technique [BR06] we have that

$$\Pr\left[\mathsf{Game}_0^P(\lambda)\right] - \Pr\left[\mathsf{Game}_1^P(\lambda)\right] \leq \Pr_{x,y}\left[(x,y) \in \mathsf{Bad}(\lambda)\right] \leq \frac{1}{2^{\mathsf{H.ol}(\lambda)/2}} \;,$$

where the last inequality holds from the union bound and the fact that $\mathsf{H.ol}(\lambda) \geq 2 \cdot \mathsf{H.kl}(\lambda)$. For this note there are at most $2^{\mathsf{H.kl}(\lambda)}$ possible values for $\mathsf{H.Ev}(1^\lambda, \mathsf{hk}, x)$. A random y would be one of the image values with probability at most $2^{\mathsf{H.kl}(\lambda)}2^{-\mathsf{H.ol}(\lambda)}$, which by assumption is at most $2^{-\mathsf{H.ol}(\lambda)/2}$.

We bound the change in P's advantage from $\mathsf{Game}_1(\lambda)$ to $\mathsf{Game}_2(\lambda)$ based on the security of the obfuscator. Let $\mathsf{Game}_i[x, y](\lambda)$ denote $\mathsf{Game}_i(\lambda)$ where the source S chooses x as its query and the $\textsc{Hash}$ oracle on the single query x chooses y as its response. We define $D[x, y](C)$ to be a distinguisher against the indistinguishability obfuscator that gets as input a circuit C, which is either an obfuscation of the zero circuit $C_0 := Z_\lambda(\cdot)$ or an obfuscation of circuit $C_1 := C_{\lambda,\mathsf{H},x,y}(\cdot)$. Distinguisher $D[x,y](C)$ runs the predictor P on C to get x', and returns $(x = x')$. Note that, according to the rules of the game (due to event Bad) both C_0 and C_1 implement the same functionality (namely zero). If $C = C_1$, then distinguisher $D[x,y]$ perfectly simulates $\mathsf{Game}_1[x,y](\lambda)$ for P. In case $D[x,y]$ gets as input $C = C_0$, it perfectly simulates $\mathsf{Game}_2(\lambda)$ for P. Hence,

$$\begin{aligned}
\Pr\left[\mathsf{Game}_1^P(\lambda)\right] - \Pr\left[\mathsf{Game}_2^P(\lambda)\right] &= \mathbb{E}_{(x,y)\notin\mathsf{Bad}(\lambda)}\left[\Pr\left[\mathsf{Game}_1^P[x,y](\lambda)\right]\right] - \mathbb{E}_{(x,y)\notin\mathsf{Bad}(\lambda)}\left[\Pr\left[\mathsf{Game}_2^P[x,y](\lambda)\right]\right] \\
&= \mathbb{E}_{(x,y)\notin\mathsf{Bad}(\lambda)}\Big[\Pr\left[\mathsf{Game}_1^P[x,y](\lambda)\right] - \Pr\left[\mathsf{Game}_2^P[x,y](\lambda)\right]\Big] \\
&= \mathbb{E}_{(x,y)\notin\mathsf{Bad}(\lambda)}\Big[\mathsf{Adv}^{\mathsf{io}}_{\mathsf{iO},D[x,y],C_0,C_1}(\lambda)\Big] \\
&\leq \max_{(x,y)\notin\mathsf{Bad}(\lambda)} \mathsf{Adv}^{\mathsf{io}}_{\mathsf{iO},D[x,y],C_0,C_1}(\lambda)\ .
\end{aligned}$$

By the security of the indistinguishability obfuscation we have that for any pair (x,y) (which is not in $\mathsf{Bad}(\lambda)$ and hence circuit $C_1 = C_{\lambda,\mathsf{H},x,y}$ is the constant zero circuit) the advantage function $\mathsf{Adv}^{\mathsf{io}}_{\mathsf{iO},D[x,y],C_0,C_1}(\lambda)$ is negligible, and hence so is the function $\max_{(x,y)\notin\mathsf{Bad}(\lambda)} \mathsf{Adv}^{\mathsf{io}}_{\mathsf{iO},D[x,y],C_0,C_1}(\lambda)$.

In game $\mathsf{Game}_2(\lambda)$ the leakage contains no information on value x and hence the probability of predictor P guessing x is upper bounded by $\mathsf{Q}(\lambda)/2^{\mathsf{H.il}(\lambda)}$, where $\mathsf{Q}(\lambda)$ denotes the number of oracle queries of S, which is negligible. Putting the above sequence of inequalities together, we get that the source is unpredictable.

Dropping the requirement. It remains to argue how we can drop the requirement on the size of hash keys. For this note that we can simply choose a t such that $t \geq 2 \cdot \lceil \mathsf{H.kl}(\lambda)/\mathsf{H.ol}(\lambda)\rceil$ and let the source leak an obfuscation of the circuit $(\mathsf{H.Ev}(1^\lambda,\cdot,x_1) = y_1 \wedge \cdots \wedge \mathsf{H.Ev}(1^\lambda,\cdot,x_t) = y_t)$. □

In the above proof, we relied on the source being able to make multiple queries to its hash oracle. Bellare, Hoang, and Keelveedhi [BHK13d] point out that the theorem can be extended to a single-query source by applying a pseudorandom generator to the output of the hash function. This result is noteworthy as several applications only require the source to make a single query.

3.2 Statistical unpredictability

The iO attack immediately gives rise to the following question: can the UCE1 and/or UCE2 notions be somehow patched so that they avoid the attack while maintaining (part of) their wide applicability? Fortunately, we show that this is indeed the case. We start by observing that the security guarantee of the indistinguishability obfuscator is only computational. Consequently, the attack can be directly ruled out by demanding the source to be *statistically* unpredictability, i.e., by letting a potential predictor run in unbounded time (but still impose polynomial query complexity). More formally, we say a source S is *statistically unpredictable* if the advantage of any (possibly unbounded) predictor P with polynomial query complexity in the $\mathsf{Pred}_S^P(\lambda)$ game shown in Figure 3 (middle) is negligible. Statistical UCE2 security can be defined analogously, where we let the reset distinguisher run in unbounded time and only place a polynomial bound on the number of its queries.

The above definition, in turn, leads to the following two questions: (1) Is a statistically secure variant of indistinguishability obfuscation possible? (2) Are there any application scenarios which only rely on this weaker property? Goldwasser and Rothblum [GR07] provide a negative answer to the first question by showing that the existence of a statistically secure iO scheme implies the collapse of the polynomial hierarchy to its second level. This impossibility result reinforces our confidence in the soundness of the above definition. For the second question, recall that the unpredictability game is always defined with respect to a random oracle, and hence statistical unpredictability may be (non-trivially) achievable.

Indeed, consider a source which samples a random point x, queries it to its oracle, and leaks the result to the distinguisher. It is easy to see that this source is statistically unpredictable as a random oracle is one-way against unbounded adversaries. Indeed, many of the cryptosystems considered by BHK admit security proofs with sources that essentially take this simple form [BHK13a, BHK13b]. We present a brief discussion of these, next.

KDM security of BRS: The unpredictability advantage of the source used in the KDM security of the BRS scheme [BRS02] is upper bounded by the probability of guessing one of polynomially many randomly chosen λ-bit strings. The probability of guessing each string is $2^{-\lambda}$, and hence the unpredictability advantage is, information theoretically, negligible.

RKA security of BRS: The unpredictability advantage is upper bounded by the number of message pairs queried to the challenge oracle times the maximum probability of guessing the output of a related-key derivation (RKD) function (on a randomly chosen key). By the assumption on the *statistical* output unpredictability of the allowed RKD functions, we get that the prediction probability is negligible.

CIH security: The unpredictability advantage here is upper bounded by the number of hash queries times the maximum probability of guessing the output of a correlated-input derivation (CID) function on a randomly chosen input point. By assumption, the CID set is statistically output unpredictable, and hence the unpredictability advantage is also negligible.

Point obfuscation: The unpredictability of the source used in the security of the hash-then-compare point obfuscation scheme is upper bounded by the maximum probability of guessing the obfuscated point α. Since α is assumed to have high min-entropy, we also recover this application.

Storage: Here we rely on the probability of guessing the stored data, which is assumed to be negligible. The unpredictability advantage is upper bounded by a polynomial multiple of this probability. Consequently the storage application also survives.

Garbling schemes: This application relies on statistically reset-secure sources. To salvage this application, we note that reset security is information theoretically upper bounded by a polynomial multiple of $2^{-\lambda}$.

After we communicated our attack [BHK13d], BHK in the revised version of their paper [BHK13c] also independently suggested the statistical notion of unpredictability. They denote by $\mathcal{S}^{\mathrm{sup}}$ the class of all statistically unpredictable sources and recast their proofs of the above to use UCE[$\mathcal{S}^{\mathrm{sup}}$]. We refer to [BHK13c] for details on the applications that can be salvaged with statistical UCE1 aka UCE[$\mathcal{S}^{\mathrm{sup}}$] (resp. statistical UCE2 aka UCE[$\mathcal{S}^{\mathrm{srs}}$]).

We end this section by noting that for the hardcore predicate, BR93 encryption, D-PKE, MLE and OAEP application scenarios discussed in [BHK13a, BHK13b], the leakage contains auxiliary information related to a query x that only computationally hides x (e.g., it might contain a one-way image $f(x)$, or an encryption of x). Consequently, an unbounded predictor might well be able to guess the point x, and in these cases our statistical patch is no longer useful. Despite this, we observe that UCE-secure hash functions with regard to statistical unpredictability are hardcore for highly non-injective one-way functions. (The proof is essentially equivalent to that in [BHK13b] and relies on the fact that any (even an unbounded) predictor cannot recover the *exact* query if the preimage space is super-polynomially large.)

```
Prl Source $S^{\textsc{Hash}}(1^\lambda)$
$(L_0, \mathbf{L}') \leftarrow\!\$\ S_0(1^\lambda)$
for $i = 1, \ldots, |\mathbf{L}'|$ do $\mathbf{L}[i] \leftarrow\!\$\ S_1^{\textsc{Hash}}(1^\lambda, \mathbf{L}'[i])$
$L \leftarrow (L_0, \mathbf{L})$
return $L$
```

```
Splt Source $S^{\textsc{Hash}}(1^\lambda)$
$(L_0, \mathbf{x}) \leftarrow\!\$\ S_0(1^\lambda)$
for $i = 1, \ldots, |\mathbf{x}|$ do $\mathbf{y}[i] \leftarrow\!\$\ \textsc{Hash}(\mathbf{x}[i])$
$L_1 \leftarrow\!\$\ S_1(1^\lambda, \mathbf{y}); L \leftarrow (L_0, L_1)$
return $L$
```

Figure 4: The parallel source $S = \mathsf{Prl}[S_0, S_1]$ on the left and the split source $S = \mathsf{Splt}[S_0, S_1]$ on the right as defined in the updated version of [BHK13c]. In both cases the source consists of two parts S_0 and S_1 that jointly generate leakage L. For split sources neither part gets direct oracle access to Hash. For parallel sources additional restrictions on the runtime and the number of queries of S_1, and the length of leakage L_0 are imposed. Note that the invocations of S_1 are parallelizable and independent of one another.

4 Bounded Parallel Sources

In the updated version of their paper [BHK13c], BHK introduce novel UCE-type security notions to recover applications where statistical unpredictability is of no help. The main idea behind these new UCE assumptions is that, in order to keep the unpredictability condition computational, the source needs to operate in a restricted way so that the iO attack cannot be mounted any longer.

A new restricted source class that BHK introduce to recover the deterministic public-key encryption (D-PKE), message-locked encryption (MLE), and OAEP applications is that of *bounded parallel sources*. In parallel sources the source splits into two parts S_0 and S_1 as follows. The first part of the source S_0 does not get oracle access to Hash, and simply outputs some preliminary leakage L_0 and a vector $\mathbf{L}'$ of arbitrary bit strings. For each entry in $\mathbf{L}'$ an independent instance of the second part of the source S_1 is run. This can be done in parallel as the several invocations do not share any coins or state. Instance i of S_1 is given $\mathbf{L}'[i]$ as input which then produces leakage $\mathbf{L}[i]$. As opposed to S_0, the second part S_1 of parallel sources has oracle access to Hash. The final leakage of the source $S := \mathsf{Prl}[S_0, S_1]$ is set to be $L := (L_0, \mathbf{L})$. The details of a parallel source $S = \mathsf{Prl}[S_0, S_1]$ are given in Figure 4 on the left.

Without any further restrictions, parallel sources are as powerful as regular sources: simply ignore S_0 and let a single S_1 generate the entire leakage. Thus, in order to circumvent the iO attack, further restrictions are necessary. To this end, BHK restrict the resources of S_0 and S_1 via polynomials τ, σ, and q as follows: (1) the running time (circuit size) of each invocation of S_1 is at most $\tau(\cdot)$; (2) each invocation of S_1 makes at most $q(\cdot)$ oracle queries; and (3) the length of initial leakage L_0 output by S_0 is at most $\sigma(\cdot)$. BHK then consider the class $\mathcal{S}^{\mathrm{prl}}_{\tau,\sigma,q}$ consisting of all parallel sources satisfying these bounds, and define UCE for computationally unpredictable, bounded parallel sources by considering $\mathrm{UCE}[\mathcal{S}^{\mathrm{cup}} \cap \mathcal{S}^{\mathrm{prl}}_{\tau,\sigma,q}]$.

For their results on D-PKE and MLE schemes, the parameters τ, σ, and q need to be fine-tuned according to the underlying encryption scheme. More precisely, BHK set q to 1 (each instance of S_1 makes a single hash query), σ to the size of a key-pair (0 in the case of MLEs), and τ to the runtime of the encryption operation plus the input and key sizes of the encryption scheme. It is easily seen that our basic attack does not fall into this class as long as the computation of the obfuscated circuit takes longer than what is granted by τ.

We start by observing that BHK's bound on the initial leakage L_0 output by the first part of the source S_0 seems to be unnecessary. Indeed, there are no restrictions on the size of the combined leakage $L = (L_0, \mathbf{L})$ nor on the length of the vector $\mathbf{L}'$ in the definition. This in turn allows a source to easily bypass the leakage bound by routing L_0 through S_1. To see this, note that the source S_0 can simply split L_0 into several smaller packets and place them into the components of vector $\mathbf{L}'$. Various instantiations of source S_1 now simply recover these packets and leak them via their own leakage.

Note that in choosing the parameters for bounded parallel sources, one has to strike a delicate balance between the complexity of obfuscating a hash function and the cost of encryption (resp. the application in question). Indeed, suppose that a bounded parallel source assumption with parameters as above is used to prove an MLE scheme secure in the standard model. Now if the complexity of the encryption scheme is high (e.g., because it is implemented based on iO [SW13] or because it includes (artificial) redundant code), then the assumption can be broken by the iO attack, as described in the previous section. Similarly, if one could reduce the complexity of obfuscating the hash function, an attack would become feasible. However, considering the current state of research, obfuscation is a very constly operation and thus, intuitively, computing the obfuscation of a hash function should be harder than encrypting a message. Interestingly, it is the *parallel* complexity of obfuscating a hash function (after a possibly complex preprocessing phase) that matters for the attack, and we can show that the latter can lie in a complexity class which is dramatically below that of computing the obfuscation of the hash function. More precisely, we show how to combine our iO attack with the *randomized encodings* of Applebaum, Ishai, and Kushilevitz [AIK04] to split the attack into two stages such that the second stage is highly parallelizable. Before describing our attack, let us briefly recall the notion of randomized encodings.

4.1 Randomized encodings

Randomized encodings allow one to substantially reduce the complexity of computing a function f by instead computing an *encoding* of it. This technique was first introduced by Ishai and Kushilevitz [IK00, IK02] in the context of multi-party computation and has since found many applications [AIK04, AIK06, IKOS08, GIS$^+$10, AIKW13, App13]. The formalization of randomized encodings that we use here is due to Applebaum, Ishai, and Kushilevitz (AIK) [AIK04] and is adapted to the setting of perfect correctness and computational privacy. Informally, we say that $\hat{f}(x; r)$ is a randomized encoding of some function $f(x)$ if (1) given $\hat{f}(x; r)$ one can efficiently recover function value $f(x)$, and (2) given $f(x)$, one can efficiently sample from the distribution $\hat{f}(x; r)$ induced by uniformly choosing r.

More precisely, a randomized encoding scheme RE consists of three efficient algorithms $(\mathsf{enc}, \mathsf{dec}, \mathsf{Sim})$ as follows: (1) a probabilistic encoding algorithm enc which on input a security parameter 1^λ, a circuit computing $f_\lambda : \{0,1\}^{n(\lambda)} \to \{0,1\}^{\ell(\lambda)}$ (of size polynomial in λ) and an $x \in \{0,1\}^{n(\lambda)}$ outputs an encoding $z \in \{0,1\}^{s(\lambda)}$; (2) a deterministic decoder algorithm dec which on input the security parameter 1^λ and an encoding $z \in \{0,1\}^{s(\lambda)}$ outputs an image point $y \in \{0,1\}^{\ell(\lambda)}$; and (3) a probabilistic simulation algorithm Sim which on input 1^λ and an image point $y \in \{0,1\}^{\ell(\lambda)}$ outputs an encoding $z \in \{0,1\}^{s(\lambda)}$. To keep our notation consistent with the previous literature on randomized encoding, for a given circuit f_λ, we will refer to the the mapping $\mathsf{enc}(1^\lambda, f_\lambda, \cdot; \cdot)$ by $\hat{f}_\lambda : \{0,1\}^{n(\lambda)} \times \{0,1\}^{m(\lambda)} \to \{0,1\}^{s(\lambda)}$, where $\{0,1\}^{m(\lambda)}$ is the randomness space of enc. We say scheme RE is a perfectly correct, computationally private randomized encoding for a circuit class $\{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ if it satisfies the following two conditions.

- Correctness. For any $f_\lambda \in \mathcal{F}_\lambda$ and any input $x \in \{0,1\}^{n(\lambda)}$ we have that

$$\Pr\left[\mathsf{dec}(1^\lambda, \hat{f}_\lambda(x; r_{\mathsf{enc}})) = f_\lambda(x) : r_{\mathsf{enc}} \leftarrow_{\$} \{0,1\}^{m(\lambda)}\right] = 1 \ .$$

- Privacy. For any PPT distinguisher D, any $f_\lambda \in \mathcal{F}_\lambda$, and any input $x \in \{0,1\}^{n(\lambda)}$ the following distinguishing advantage is negligible:

$$\mathsf{Adv}^{\mathsf{re}}_{\mathsf{RE},D,x}(\lambda) := \Pr[D(1^\lambda, \hat{f}_\lambda(x; r_{\mathsf{enc}})) = 1 : r_{\mathsf{enc}} \leftarrow_{\$} \{0,1\}^{m(\lambda)}] - \Pr[D(1^\lambda, \mathsf{Sim}(1^\lambda, f_\lambda(x)) = 1] \ .$$

Functions n, ℓ, s, and m are polynomials, however, we will be dropping the explicit dependency on λ in order to simplify notation, and set $n := n(\lambda)$, $\ell := \ell(\lambda)$, $s := s(\lambda)$, and $m := m(\lambda)$.

A randomized encoding can be trivially achieved by setting $\hat{f}(x; r) := f(x)$. However one is usually interested in an encoding, $\hat{f}$, which is in a low complexity class (typically $\mathcal{NC}^0$), although f itself might be only computable in a higher class. AIK, utilizing garbled circuits, construct such encodings for several standard cryptographic primitives, such as one-way functions and pseudo-random generators, in $\mathcal{NC}^0$. In our case, however, the complexity of the encoding is not crucial and, indeed, we only require it to be computable in polynomial time. Rather, for our application, we are concerned with the *input locality* of the computation of $\hat{f}$. More precisely, we want the number of input bits of x which affect each output bit of the encoding $\hat{f}(x;r)$ to be small. We will return to the topic of locality in Section 4.3.

4.2 Composing iO with randomized encodings

To ease readability, we present our attack in two stages. First, we show that our iO attack can be composed with any randomized encoding scheme in a way which neither affects the adversary's advantage nor the unpredictability of its implicit source. Then, in the next subsection, we use a special type of RE scheme known as *decomposable randomized encodings* [IKOS08] to split and parallelize the adversary's source in order to meet the (minimal) bounds of $q(\lambda) = 1$, $\sigma(\lambda) = 0$, and $\tau(\lambda) \in \mathcal{O}(\lambda)$. Consequently, our attack will rule out bounded parallel sources for these parameters. Since the bounds that our attacks achieves are very stringent, and an encryption scheme has to at least run in time $\mathcal{O}(\lambda)$ (and make a single Hash query), assuming indistinguishability obfuscation, it is unlikely that bounded parallel sources can be used to instantiate ROs in any meaningful application scenario.

Let H be a UCE$[\mathcal{S}^{\mathrm{cup}} \cap \mathcal{S}^{\mathrm{prl}}_{\tau,\sigma,q}]$-secure hash function, iO be an indistinguishability obfuscator, and let us assume once again that $\mathsf{H.ol}(\lambda) \geq 2 \cdot \mathsf{H.kl}(\lambda)$. (As in the proof of Theorem 3.1, this assumption will be without loss of generality.)

The attacker. Define $C_{\lambda,\mathsf{H},x,y}(\cdot) := (\mathsf{H.Ev}(1^\lambda, \cdot, x) = y)$, and compute a randomized encoding of the circuit

$$f : (x, y, r_{\mathsf{io}}) \mapsto \mathsf{iO}(C_{\lambda,\mathsf{H},x,y}(\cdot); r_{\mathsf{io}}),$$

where r_{io} is the randomness used by the obfuscator. As in the proof of Theorem 3.1, we consider the source S which chooses random values x, r_{io}, and r_{enc}, queries x to its oracle to obtain $y \leftarrow \textsc{Hash}(x)$, and leaks the randomized encoding

$$L := \hat{f}(x, y, r_{\mathsf{io}}; r_{\mathsf{enc}}) \ .$$

The distinguisher D gets as input a hash key hk and an encoding $\hat{f}(x, y, r_{\mathsf{io}}; r_{\mathsf{enc}})$. It uses the decoder dec of the randomized encoding scheme to recover

$$if(x, y, r_{\mathsf{io}}) \leftarrow \mathsf{dec}(\hat{f}(x, y, r_{\mathsf{io}}; r_{\mathsf{enc}})) \ .$$

It then interprets the result as a circuit, runs it on on hk, and returns whatever the circuit outputs.

Advantage. The output of the decoder above, due to the definition of f and the perfect correctness of the encoding scheme, is equal to $\mathsf{iO}(C_{\lambda,\mathsf{H},x,y}(\cdot); r_{\mathsf{io}})$. The same analysis as in Theorem 3.1 shows that in case y was computed as $\mathsf{H.Ev}(1^\lambda, \mathsf{hk}, x)$, the obfuscated circuit, and hence D, always return 1. In case that y was drawn at random, it is highly unlikely that it equals $\mathsf{H.Ev}(1^\lambda, \mathsf{hk}, x)$, and thus, D returns 0 with overwhelming probability. Hence D distinguishes the real hash function H (with a randomly chosen key) from an ideal hash function with overwhelming probability. (See the proof of Theorem 3.1 on page 7 for further details.)

Unpredictability. Next we show that S is computationally unpredictable; that is, we show that x is not leaked when Hash implements a random oracle. As in Theorem 3.1, we observe that the circuit $C_{\lambda,\mathsf{H},x,y}(\cdot)$ is the zero circuit with overwhelming probability if y is chosen uniformly at random. (This is because, as before, $2|\mathsf{hk}|$ is assumed to be smaller than $|y|$ and a counting argument applies.) Let P be a predictor that succeeds with non-negligible probability in the $\mathsf{Pred}^P_S(\lambda)$ game. We bound the success probability of P based on the security of the indistinguishability obfuscator iO and the privacy of the randomized encoding scheme via a sequence of 4 games as follows.

$\mathsf{Game}_0(\lambda)$**:** is identical to the computational unpredictability game $\mathsf{Pred}_S^P(\lambda)$ (see Figure 3).

$\mathsf{Game}_1(\lambda)$**:** is similar to the previous game except that the source S now computes the leakage by first computing the circuit $\mathsf{iO}(C_{\lambda,\mathsf{H},x,y}(\cdot))$ as an obfuscation of $C_{\lambda,\mathsf{H},x,y}(\cdot)$ and then running the RE simulator Sim on it in, i.e., $L \leftarrow\!\!\$\ \mathsf{Sim}(\mathsf{iO}(C_{\lambda,\mathsf{H},x,y}(\cdot)))$.

$\mathsf{Game}_2(\lambda)$**:** is similar to the previous game except that the game terminates if there exists an $\mathsf{hk} \in \{0,1\}^{\mathsf{H.kl}(\lambda)}$ such that $C_{\lambda,\mathsf{H},x,y}(\mathsf{hk}) = 1$. We write $(x,y) \in \mathsf{Bad}(\lambda)$ if for values (x,y) such an hk exists for which $C_{\lambda,\mathsf{H},x,y}(\mathsf{hk}) = 1$.

$\mathsf{Game}_3(\lambda)$**:** is similar to the previous game except that S now computes $\mathsf{iO}(Z_\lambda(\cdot))$, where $Z_\lambda(\cdot)$ denotes the constant zero circuit, and then runs the RE simulator Sim on it in order to compute L.

We bound the difference between P's advantage in $\mathsf{Game}_0(\lambda)$ and in $\mathsf{Game}_1(\lambda)$ using the privacy of the RE scheme. We let $\mathsf{Game}_i[x,y,r_{\mathsf{io}}](\lambda)$ denote $\mathsf{Game}_i(\lambda)$ where the source S chooses x as the query point and r_{io} as coins for the obfuscator, and the HASH oracle chooses y. Let $D[x,y,r_{\mathsf{io}}](L)$ be a distinguisher that on input $L = \hat{f}(x,y,r_{\mathsf{io}};r_{\mathsf{enc}})$ or $L \leftarrow\!\!\$\ \mathsf{Sim}(\mathsf{iO}(C_{\lambda,\mathsf{H},x,y}(\cdot)))$ runs the predictor P on L to get a point x' and returns $(x = x')$. Note that when $D[x,y,r_{\mathsf{io}}]$ gets a randomized encoding, it runs the predictor in the $\mathsf{Game}_0[x,y,r_{\mathsf{io}}](\lambda)$ environment, and when it gets a simulated encoding, it runs P according to the rules of $\mathsf{Game}_1[x,y,r_{\mathsf{io}}](\lambda)$. Hence,

$$\begin{aligned}
\Pr\left[\mathsf{Game}_0^P(\lambda)\right] - \Pr\left[\mathsf{Game}_1^P(\lambda)\right] &= \mathbb{E}_{x,y,r_{\mathsf{io}}}\left[\Pr\left[\mathsf{Game}_0^P[x,y,r_{\mathsf{io}}](\lambda)\right]\right] - \mathbb{E}_{x,y,r_{\mathsf{io}}}\left[\Pr\left[\mathsf{Game}_1^P[x,y,r_{\mathsf{io}}](\lambda)\right]\right] \\
&= \mathbb{E}_{x,y,r_{\mathsf{io}}}\Big[\Pr\left[\mathsf{Game}_0^P[x,y,r_{\mathsf{io}}](\lambda)\right] - \Pr\left[\mathsf{Game}_1^P[x,y,r_{\mathsf{io}}](\lambda)\right]\Big] \\
&= \mathbb{E}_{x,y,r_{\mathsf{io}}}\Big[\mathsf{Adv}^{\mathsf{re}}_{\mathsf{RE},D[x,y,r_{\mathsf{io}}],(x,y,r_{\mathsf{io}})}(\lambda)\Big] \\
&\leq \max_{x,y,r_{\mathsf{io}}} \mathsf{Adv}^{\mathsf{re}}_{\mathsf{RE},D[x,y,r_{\mathsf{io}}],(x,y,r_{\mathsf{io}})}(\lambda)\ .
\end{aligned}$$

Furthermore, by the computational privacy of scheme RE, we have that for any (x,y,r_{io}) the advantage $\mathsf{Adv}^{\mathsf{re}}_{\mathsf{RE},D[x,y,r_{\mathsf{io}}],(x,y,r_{\mathsf{io}})}(\lambda)$ is negligible, and hence so is the function $\max_{x,y,r_{\mathsf{io}}} \mathsf{Adv}^{\mathsf{re}}_{\mathsf{RE},D[x,y,r_{\mathsf{io}}],(x,y,r_{\mathsf{io}})}(\lambda)$.

By the fundamental lemma of the game-playing technique we have that

$$\Pr\left[\mathsf{Game}_0^P(\lambda)\right] - \Pr\left[\mathsf{Game}_1^P(\lambda)\right] \leq \Pr_{x,y}\left[(x,y) \in \mathsf{Bad}(\lambda)\right] \leq \frac{1}{2^{\mathsf{H.ol}(\lambda)/2}}\ ,$$

where the last inequality holds from the union bound and the fact that $\mathsf{H.ol}(\lambda) \geq 2 \cdot \mathsf{H.kl}(\lambda)$ (see the proof of Theorem 3.1).

We bound the change in P's advantage from $\mathsf{Game}_2(\lambda)$ to $\mathsf{Game}_3(\lambda)$ based on the security of the obfuscator (this is essentially equivalent to the second game hop in the proof of Theorem 3.1). We define $\mathsf{Game}_i[x,y](\lambda)$ similarly to the previous analysis to denote the game where S chooses x as the query point and the oracle chooses y. We let $D[x,y](C)$ be a distinguisher that gets as input a circuit C, which is either an obfuscation of the zero circuit $C_0 := Z_\lambda(\cdot)$ or $C_1 := C_{\lambda,\mathsf{H},x,y}(\cdot)$. Distinguisher $D[x,y](C)$ uses the simulator Sim of the RE scheme to obtain $C' \leftarrow\!\!\$\ \mathsf{Sim}(C)$, runs the predictor P on C' to get x', and returns $(x = x')$. Note that, according to the rules of the game (due to event Bad) both C_0 and C_1 implement the same functionality (namely zero). If $C = C_1$, then distinguisher $D[x,y]$ perfectly simulates $\mathsf{Game}_2[x,y](\lambda)$ for P. In case $D[x,y]$ gets as input $C = C_0$, it perfectly simulates $\mathsf{Game}_3(\lambda)$ for P. Hence, using an analysis similar to that given above, we get

$$\Pr\left[\mathsf{Game}_2^P(\lambda)\right] - \Pr\left[\mathsf{Game}_3^P(\lambda)\right] \leq \max_{(x,y)\notin\mathsf{Bad}(\lambda)} \mathsf{Adv}^{\mathsf{io}}_{\mathsf{iO},D[x,y],C_0,C_1}(\lambda)\ .$$

By the security of iO each term $\mathsf{Adv}^{\mathsf{io}}_{\mathsf{iO},D[x,y],C_0,C_1}(\lambda)$ is negligible, and hence so is the above bound.

Finally, in $\mathsf{Game}_3(\lambda)$, the probability that P guesses x is upper bounded by $\mathsf{Q}(\lambda)/2^{\mathsf{H.il}(\lambda)}$, where $\mathsf{Q}(\lambda)$ denotes the number of oracle queries of S, which is negligible. This is because x is chosen uniformly at random and the leakage that the predictor receives is independent of x. Putting the above equations together, we get that

$$\Pr\left[\,\mathsf{Pred}_S^P(\lambda)\right] \leq \max_{x,y,r_{\mathsf{io}}} \mathsf{Adv}^{\mathsf{re}}_{\mathsf{RE},D[x,y,r_{\mathsf{io}}],(x,y,r_{\mathsf{io}})}(\lambda) + \max_{(x,y)\notin\mathsf{Bad}(\lambda)} \mathsf{Adv}^{\mathsf{io}}_{\mathsf{iO},D[x,y],C_0,C_1}(\lambda) + \frac{1}{2^{\mathsf{H.ol}(\lambda)/2}} + \frac{\mathsf{Q}(\lambda)}{2^{\mathsf{H.il}(\lambda)}} \ .$$

Therefore P's advantage is negligible, and the source S is computationally unpredictable.

4.3 Splitting and parallelizing S using decomposable REs

Our analysis in the previous section holds for any randomized encoding scheme. We show that if a *decomposable* scheme, is used to instantiate the attack, we recast the source above as a bounded parallel source. Let us begin with the definition decomposable randomized encodings.

Decomposable encodings. In a decomposable randomized encoding (DRE) scheme, every output bit of the encoding $\hat{f}(x;r)$ depends on at most a single bit of x (but possibly on arbitrarily many bits of r). More precisely, a decomposable randomized encoding scheme DRE consists of a four tuple of algorithms $(\mathsf{idx}, \mathsf{enc}, \mathsf{dec}, \mathsf{Sim})$ as follows. Algorithm idx on input a circuit f and an index $i \in [s]$ outputs an index $j \in [n] \cup \{0\}$. The decomposable encoding algorithm enc operates based on a local encoding algorithm $\overline{\mathsf{enc}}$ as follows. On input a circuit f, a point x, and random coins r_{enc}, for each $i \in [s]$ compute $z_i \leftarrow \overline{\mathsf{enc}}(f, i, x[\mathsf{idx}(f,i)]; r_{\mathsf{enc}})$, where we define $x[0] :=\perp$, and return $z \leftarrow (z_1, \ldots, z_s)$. Algorithms dec and Sim play the same roles as those in a conventional RE scheme. As before, we denote $\overline{\mathsf{enc}}(f, i, b; r_{\mathsf{enc}})$ by $\hat{f}_i(b; r_{\mathsf{enc}})$. Thus we may write

$$\hat{f}(x; r_{\mathsf{enc}}) = \hat{f}_1(x[\mathsf{idx}(1)]; r_{\mathsf{enc}}) \| \hat{f}_2(x[\mathsf{idx}(2)]; r_{\mathsf{enc}}) \| \cdots \| \hat{f}_s(x[\mathsf{idx}(s)]; r_{\mathsf{enc}}) \ .$$

As Ishai et al. [IKOS08] point out, several constructions of randomized encodings are decomposable. For example, AIK's construction based on garbled circuits [AIK06] is a decomposable, perfectly correct, and computationally private randomized encoding for any function in $\mathcal{P}/poly$. Their construction relies only on the existence of secure pseudorandom generators.

We show that we can compute decomposable randomized encodings in a very special way consisting of two phases where the first phase is not given the entire input (this can easily be extended such that the first phase does not depend on the input at all), and the second phase can be computed in parallel by constant-depth circuits of low gate count. We will use this result to adapt our above attack to bounded parallel sources by constructing a source $S = \mathsf{Prl}[S_0, S_1]$ that computes the very same encoding that our previously described source did. Looking ahead, we note that the decomposability of the encoding is convenient because every bit of the encoding depends on only a single bit of the actual input and one can easily precompute both possibilities (that is, the encodings when the input bit is 0 and when it is 1) as long as one knows the randomness for the encoding. Once the actual input is known all one has to do to compute a proper encoding is to drop one of the two bits, an operation which can be easily parallelized.[2]

Algorithm S_0. Algorithm S_0 of the source (see Figure 5) begins by generating the randomness for the randomized encoder and the obfuscator (line 2). It then picks a random x in the domain of the hash function and sets an auxiliary variable aux to $x \| 0^{\mathsf{H.ol}(\lambda)} \| r_{\mathsf{io}}$ (line 4). We will use this variable to access the specific input bits that the randomized encoder needs. Note that we have not yet specified a

[2] The (decomposable) randomized encoding of [AIK06] is in $\mathcal{NC}^0$ assuming that pseudorandom generators exist in $\mathcal{NC}^0$. Thus, using this randomized encoding scheme, the pre-computation part can also be parallelized.

```
Algo. $S_0(1^\lambda)$
1  $\mathbf{L}' \leftarrow []$
2  $r_{\mathsf{enc}} \leftarrow\$ \{0,1\}^{\mathsf{DRE.rl}(\lambda)}$;  $r_{\mathsf{io}} \leftarrow\$ \{0,1\}^{\mathsf{iO.rl}(\lambda)}$
3  $x \leftarrow\$ \{0,1\}^{\mathsf{H.il}(\lambda)}$
4  $\mathsf{aux} \leftarrow x \| 0^{\mathsf{H.ol}(\lambda)} \| r_{\mathsf{io}}$
5  for $i = 1 \ldots s$ do
6      if $\mathsf{H.il}(\lambda) < \mathsf{idx}(i) \leq \mathsf{H.il}(\lambda) + \mathsf{H.ol}(\lambda)$ then
7          $j \leftarrow \mathsf{idx}(i) - \mathsf{H.il}(\lambda)$
8          $b_0 \leftarrow \hat{f}_i(0; r_{\mathsf{enc}})$
9          $b_1 \leftarrow \hat{f}_i(1; r_{\mathsf{enc}})$
10         $\mathbf{L}'[i] \leftarrow 1 \| b_0 \| b_1 \| j \| x$
11     else
12         $b \leftarrow \hat{f}_i(\mathsf{aux}[\mathsf{idx}(i)]; r_{\mathsf{enc}})$
13         $\mathbf{L}'[i] \leftarrow 0 \| b$
14 return $(\varepsilon, \mathbf{L}')$
```

```
Algo. $S_1^{\text{Hash}}(\mathbf{L}'[i])$
$c \leftarrow \text{MostSigBit}(\mathbf{L}'[i])$
if $c = 0$ then
    parse $\mathbf{L}'[i]$ as $0 \| b$
    $L \leftarrow b$
else
    parse $\mathbf{L}'[i]$ as $1 \| b_0 \| b_1 \| j \| x$
    $y \leftarrow \text{Hash}(x)$
    if $y[j] = 0$ then
        $L \leftarrow b_0$
    else
        $L \leftarrow b_1$
return $L$
```

Figure 5: Pseudocode of the parallel source $S = \mathsf{Prl}[S_0, S_1]$.

proper value for y at this point, but fixed it to the arbitrary string $0^{\mathsf{H.ol}(\lambda)}$. Next, algorithm S_0 generates a leakage value for every output bit of the encoding, i.e., for every $i \in [s]$ (where s is the size of the encoding). Here, we distinguish two cases. If the i-th output bit of the encoding depends on an input bit corresponding to y, then the if branch in line 6 is executed. Otherwise, the else branch in line 11 is run. In the first case, we first compute the index j of y that the encoding depends upon (line 7). As at this point y has not yet been chosen, S_0 computes the output bits for the two possible values of $y[j]$; that is, it stores the output bit for $y[j] = 0$ as b_0 and the output bit for $y[j] = 1$ as b_1. It then sets the leakage at position i to $1 \| b_0 \| b_1 \| j \| x$ (line 10). In the second case (i.e., when the output bit of the encoding does not depend on a bit of y), algorithm S_0 simply computes the output bit (it knows the value of the corresponding input bit), stores it in b, and sets the leakage at position i to $0 \| b$ (line 13). At the end of the for loop $\mathbf{L}'$ contains a single value for every output bit of the randomized encoding. Algorithm S_0 returns $\mathbf{L}'$.

Algorithm S_1. An independent instance of algorithm S_1 (see Figure 5) for every entry in $\mathbf{L}'$ is run. On input $\mathbf{L}'[i]$ algorithm S_1 checks if its input is of the form $0 \| b$. If so, it simply sets $L \leftarrow b$ and returns L. Else, it parses $\mathbf{L}'[i]$ as $1 \| b_0 \| b_1 \| j \| x$ and computes $y \leftarrow \text{Hash}(x)$. (Note that S_1 has access to Hash.) It then sets $L \leftarrow b_0$ if $y[j] = 0$ and $L \leftarrow b_1$ otherwise. Finally, it outputs L.

By construction, the parallel source $S := \mathsf{Prl}[S_0, S_1]$ computes the same randomized encoding $\hat{f}(x, y, r_{\mathsf{io}}; r_{\mathsf{enc}})$ that our previous source did. Consequently, by setting the distinguisher D to be identical to that given in the previous attack, we obtain a successful bounded parallel attack. Furthermore, (each instance of) algorithm S_1 makes a single call to Hash and can be implemented by a constant-depth circuit. Finally, and consistently with our observation that L_0 can be routed via the second stage, algorithm S_0 always returns $L_0 = \varepsilon$.

Dropping the length requirement. As in Theorem 3.1, in order to drop the requirement that $\mathsf{H.ol}(\lambda) \geq 2 \cdot \mathsf{H.kl}(\lambda)$ we may choose a $t \geq \lceil \mathsf{H.kl}(\lambda)/\mathsf{H.ol}(\lambda) \rceil$ and use the circuit

$$C_{\lambda, \mathsf{H}, (x_1,y_1), \ldots, (x_t,y_t)}(\cdot) := (\mathsf{H.Ev}(1^\lambda, \cdot, x_1) = y_1 \wedge \cdots \wedge \mathsf{H.Ev}(1^\lambda, \cdot, x_t) = y_t) \ .$$

When applying the decomposable randomized encoding scheme, each instance of S_1 is responsible to compute exactly one output bit of the encoding. As each output bit depends on at most a single input bit, it can depend on at most one of the y_k's (for $k = 1, \ldots, t$). We adapt algorithm S_0 to prepare the values $\mathbf{L}'[i] \leftarrow 1 \| b_0 \| b_1 \| j_k \| x_k$ and, thus, can leave algorithm S_1 unchanged.

Putting the above together, we obtain the following result.

Theorem 4.1 (Bounded parallel UCE infeasibility). *If indistinguishability obfuscation (and PRGs) exist, then* $\mathrm{UCE}[\mathcal{S}^{\mathrm{cup}} \cap \mathcal{S}^{\mathrm{prl}}_{\tau,\sigma,q}]$ *security cannot be achieved in the standard model for* $q \neq 0$*, any* $\sigma \geq 0$*, and* $\tau \in \Omega(\lambda)$*.*

Specifying the size of S_1. We can specify the size of a circuit that computes S_1, thereby giving a precise lower bound on τ. For this we encode index j (the bit position of y the encoding depends on) by a bitmap. That is, we encode it by a bit string y_{idx} of length $|y_{\mathsf{idx}}| = \mathsf{H.ol}(\lambda)$ that contains a single 1 at the j-th position:

$$y_{\mathsf{idx}} := 0^{j-1} \| 1 \| 0^{\mathsf{H.ol}(\lambda)-j} \ .$$

Furthermore, we let the output of S_1 to be a bit string of length $1 + \mathsf{H.ol}(\lambda)$ rather than a single bit. This string will contain at most a single 1 and hence, in order to recover the correct output bit, the distinguisher must simply compute the logical OR of all the bits in the string.

Consider the following input to S_1

$$d \| b_0 \| b_1 \| y_{\mathsf{idx}}[1] \| \ldots \| y_{\mathsf{idx}}[\mathsf{H.ol}(\lambda)] \| x[1] \| \ldots \| x[\mathsf{H.ol}(\lambda)] \ ,$$

where we assume that in case it does not depend on y (see line 13), it is padded with zeros. The circuit computes values $(y[1], \ldots, y[\mathsf{H.ol}(\lambda)]) \leftarrow \textsc{Hash}(x)$, and outputs

$$\begin{gathered} \neg d \wedge b_0 \ , \\ y_{\mathsf{idx}}[1] \wedge \Big((\neg y[1] \wedge b_0) \vee (y[1] \wedge b_1) \Big) \ , \\ \vdots \\ y_{\mathsf{idx}}[\mathsf{H.ol}(\lambda)] \wedge \Big((\neg y[\mathsf{H.ol}(\lambda)] \wedge b_0) \vee (y[\mathsf{H.ol}(\lambda)] \wedge b_1) \Big) \ . \end{gathered}$$

The first bit computed above corresponds to line 4 of algorithm S_1 (see Figure 5). The remaining bits correspond to the computation of the else clause (line 5). Each of these bits corresponds to testing whether the t-th bit of y needs to be considered (that is, if $j = t$; see line 7 of algorithm S_0) and whether to output b_0 or b_1 if this is the case. Note that, by construction, at most a single bit of the output is set to 1, and to recover the output of our original algorithm S_1 all that the distinguisher needs to do is to compute the logical OR of the bits of the string. Furthermore, we note that delegating the computation of the final OR to the distinguisher does not leak any information, since, given the result of the OR operation, the distinguisher can reconstruct the bit string: If the resulting bit is 0, then the bit string was the all-zero string. Else, the distinguisher only has to evaluate the index function idx of the randomized encoding to reconstruct the position of the single 1 within the bit string.

By counting the operations above, we get that S_1 can be implemented with a single oracle gate, $\mathsf{H.ol}(\lambda)$ many OR gates, $\mathsf{H.ol}(\lambda) + 1$ many NOT gates, and $1 + 3 \cdot \mathsf{H.ol}(\lambda)$ AND gates where all the AND and OR gates have fan-in 2. Thus, our implementation of S_1 is in $\mathcal{NC}^0$.

5 Split Sources: A Discussion

In their original paper [BHK13a], BHK show that a UCE1-secure hash function is sufficiently strong to produce simultaneous hardcore bits for any one-way function f and replace the random oracle in the BR93 encryption scheme. As UCE1-secure hash functions are prone to our iO attack, in the updated version of their paper [BHK13c], BHK introduce a new class of sources, called *split sources*, to salvage these applications. A split source S is composed of two algorithms S_0 and S_1 as follows. Neither algorithm gets access to the $\textsc{Hash}$ oracle. Algorithm S_0 outputs L_0 together with a vector of points $\mathbf{x}$. For each entry of $\mathbf{x}$, the corresponding Hash value is computed, and the vector of hash values $\mathbf{y}$ is formed. Algorithm S_1 is then run on $\mathbf{y}$, and L_1 is produced. The leakage of the split source $S := \mathsf{Splt}[S_0, S_1]$ is set to $L := (L_0, L_1)$ (which is then passed to the distinguisher). The pseudocode for split sources is give in Figure 4 on the right. We say that a source S is in class $\mathcal{S}^{\text{splt}}$ if there exists PPT algorithms S_0 and S_1 such that $S = \mathsf{Splt}[S_0, S_1]$. It is then shown that if $\mathsf{H} \in \text{UCE}[\mathcal{S}^{\text{cup}} \cap \mathcal{S}^{\text{splt}}]$, that is, if H is UCE-secure with respect to split, computationally unpredictable sources, then H is hardcore for any one-way function f and can be also securely used in the BR93 encryption scheme.

Both values x and y in the construction of the circuit $\mathsf{H.Ev}(1^\lambda, \cdot, x) = y$ are needed. However, since there is no direct communication between the two components of a split source, and since each component only sees one of these values, this circuit cannot be formed, and hence the iO attack cannot be launched. The generalized attack from Section 4 also falls outside this source class since there is no direct communication between the two source components.

In this section we discuss some intricacies of such sources. We show that when composing $\text{UCE}[\mathcal{S}^{\text{cup}} \cap \mathcal{S}^{\text{splt}}]$ secure hash functions with one-way permutations, then the resulting function is not $\text{UCE}[\mathcal{S}^{\text{cup}} \cap \mathcal{S}^{\text{splt}}]$ secure. We go on to present necessary conditions for the security of such $\text{UCE}[\mathcal{S}^{\text{cup}} \cap \mathcal{S}^{\text{splt}}]$ secure functions. Although our observations are presented for split sources, due to their simplicity, they may apply to other notions of computational unpredictability as well.

5.1 Composing UCEs with one-way permutations

Let H be a hash function and let π be a one-way permutation with appropriate domain and range. Consider the function H' defined by $\mathsf{H}'.\mathsf{Ev}(1^\lambda, \mathsf{hk}, x) := \mathsf{H.Ev}(1^\lambda, \mathsf{hk}, \pi(x))$. That is, the input x is first run through a one-way permutation before being hashed. Intuitively, this application of a one-way permutation should not harm the security of H'. Indeed, this can be easily seen to be the case for the one-wayness, collision resistance, and pseudorandomness properties. We show that statistical UCE1 also enjoys this property.

Proposition 5.1. *Let H, π, and H' be as defined above. Suppose that $\mathsf{H} \in \text{UCE}[\mathcal{S}^{\text{sup}}]$. Then we also have that $\mathsf{H}' \in \text{UCE}[\mathcal{S}^{\text{sup}}]$.*

Proof. Given an adversary (S', D') against the $\text{UCE}[\mathcal{S}^{\text{sup}}]$ security of H', we construct an adversary (S, D) against the $\text{UCE}[\mathcal{S}^{\text{sup}}]$ security of H as follows. Source S runs S' and answers each oracle query x by computing $\pi(x)$, querying its own Hash oracle on $\pi(x)$, and returning the answer to S'. When S' outputs some leakage L, algorithm S also outputs L as its own leakage. The distinguisher D is defined to be identical to D'. Note that (S, D) runs (S', D') in an environment identical to the UCE game for H', and (S, D) succeeds in guessing the correct bit with the same probability that (S', D') does. It remains to show that S is statistically unpredictable. This follows by observing that any predictor P which successfully guesses a query $\pi(x)$ of S can be transformed into an algorithm P' which runs in *unbounded* time and also returns x. This is a query of S, and this contradicts the statistical unpredictability of S'. □

Somewhat counterintuitively, we show that composition provably does not hold for split sources.

Proposition 5.2. *Let H, π, and H' be as defined above. Then $\mathsf{H}' \notin \text{UCE}[\mathcal{S}^{\text{cup}} \cap \mathcal{S}^{\text{splt}}]$ (irrespectively of the security guarantees of H).*

Proof. Consider the following (split) source $S := \mathsf{Splt}[S_0, S_1]$. Algorithm S_0 chooses a value x at random, computes $x' \leftarrow \pi(x)$ and outputs (x', x). Algorithm S_1 receives as input $y \leftarrow \text{Hash}(x)$ and outputs y. The final leakage is, therefore, (x', y). It is easy to see that the this source is computationally unpredictable as neither $x' = \pi(x)$ nor y allow a computationally bounded predictor to recover x. Indeed, in the unpredictability experiment (see Figure 3) y is drawn at random and independently of x and if a predictor could recover x from $(\pi(x), y)$, one could turn it into an adversary against the one-wayness of π.

Given the above source S, a distinguisher D can easily win the UCE game as follows. Algorithm D computes $y' \leftarrow \mathsf{H.Ev}(1^\lambda, \mathsf{hk}, x')$ and outputs $(y' = y)$. Note that as $\mathsf{H.Ev}(1^\lambda, \mathsf{hk}, x') = \mathsf{H'.Ev}(1^\lambda, \mathsf{hk}, x)$ the distinguisher will always output 1 in case Hash implements the real hash function $\mathsf{H'}$. On the other hand it will output 1 with only a negligible probability in case Hash implements a random oracle. □

The simplicity of the technique allows this result to be applied more generally to other computational notions of UCE security, as long as the structure of the hash function follows the one described above. This, in particular, is the case for bounded parallel sources. Note also that we did not rely on any obfuscation schemes.

5.2 One-way obfuscators vs. computational UCE security

The aforementioned attack only applies to hash functions that have a particular structure (i.e., those which first apply a one-way permutation to their inputs). Although this assumption can be somewhat relaxed—a one-way function, for example, would suffice—an attack on this special class of hash functions does not allow for any general conclusions on split source UCE security. In the following, we will discuss notions of obfuscation that might allow for a general break. Let H be a hash function. Consider the following UCE adversary (S, D). Source S chooses a random x constructs the circuit $C(\cdot) := \mathsf{H.Ev}(1^\lambda, \cdot, x)$. It then calls Hash on x to receive $y \leftarrow \textsc{Hash}(x)$ and outputs $(\mathcal{O}(C), y)$ as L, where $\mathcal{O}$ is a special purpose obfuscator that we shall be describing shortly. Distinguisher D outputs $(\mathcal{O}(C)(\mathsf{hk}) = y)$. It is easily seen that adversary (S, D) wins the UCE game with overwhelming probability.

We need to ensure that S is a split source and computationally unpredictable. The split condition can be easily seen to hold from the definition of algorithm S. The unpredictability property, however, depends on the obfuscator $\mathcal{O}$, and in particular on whether $\mathcal{O}(C)$ reveals x. In other words, the (randomized) function

$$f : x \mapsto \mathcal{O}(\mathsf{H.Ev}(1^\lambda, \cdot, x))$$

needs to be one-way. We call such a primitive a *one-way obfuscator*. A direct consequence is that in order to prove that a function H belongs to $\mathrm{UCE}[\mathcal{S}^{\mathrm{cup}} \cap \mathcal{S}^{\mathrm{splt}}]$, one needs to show that it is not one-way obfuscatable.

It is possible to construct hash functions that are provably not one-way obfuscatable in the above sense. Let H be a hash function, and define a modified function $\mathsf{H'}$ which on key $\mathsf{hk} = 0^{\mathsf{H.kl}(\lambda)}$ outputs x. That is, $\mathsf{H'.Ev}(1^\lambda, \mathsf{hk}, x) := \mathsf{H.Ev}(1^\lambda, \mathsf{hk}, x)$ if $\mathsf{hk} \neq 0^{\mathsf{H.kl}(\lambda)}$ and $\mathsf{H'.Ev}(1^\lambda, \mathsf{hk}, x) := x$ otherwise. As the security of UCE notions is defined over a random choice of key this modification does not harm UCE security. Now, given an obfuscation of the circuit $C(\cdot) := \mathsf{H'.Ev}(1^\lambda, \cdot, x)$ it is easy to recover x via a query on the *weak key* $\mathsf{hk} = 0^{\mathsf{H.kl}(\lambda)}$.

As this notion of one-way obfuscation is not achievable, let us relax the notion further and transform the attack into one that only relies on a weaker notion of one-way obfuscation. Consider the circuit $C(\cdot, \cdot) := (\mathsf{H.Ev}(1^\lambda, \cdot, x) = \cdot)$; that is, the circuit which takes two inputs hk and y and outputs 1 if $\mathsf{H.Ev}(1^\lambda, \mathsf{hk}, x) = y$, and 0 otherwise. Now, consider an S which leaks a one-way obfuscation of the circuit $C(\cdot, \cdot)$ together with the value $y = \textsc{Hash}(x)$. A distinguisher D on input hk, the obfuscated circuit, and y, can win the UCE game by computing $C(\mathsf{hk}, y)$ and directly outputting the result. Hence, in order to prove that a function H is in $\mathrm{UCE}[\mathcal{S}^{\mathrm{cup}} \cap \mathcal{S}^{\mathrm{splt}}]$, one now needs to show that

$$f : x \mapsto \mathcal{O}(\mathsf{H.Ev}(1^\lambda, \cdot, x) = \cdot)$$

is not one way. Indeed, the previous counterexample using a single "weak key" against one-way obfuscation no longer works as one would need to guess x in order to be able to extract x. However, hash functions can still be tweaked to avoid this attack. Towards demonstrating this, we will introduce a *larger* set of weak hash keys. Let H be a hash function as above, and define a modified function H' which works similarly to H except that if a key is of the form $0^i\|1\|0^j$ for $i, j \geq 0$ and $i + j = \mathsf{H.kl}(\lambda) - 1$ it returns $x[i+1]\|0^{\mathsf{H.ol}(\lambda)-1}$. It is easily established that using $\mathcal{O}(\mathsf{H}'.\mathsf{Ev}(1^\lambda, \cdot, x) = \cdot)$, the point x can be extracted with $\mathsf{H.il}(\lambda)$ many queries as the weak keys allow one to learn x bit by bit.

It is conceivable that weaker forms of obfuscation, e.g., an *approximate* version where the obfuscated circuit may err on a portion of input pairs (hk, y), might well be within the reach and powerful enough to mount an attack on UCEs even in the presence of weak keys. Furthermore, artificially introducing weaknesses (such as the above weak keys) into existing functions is not sufficient unless the function is changed on a large fraction of keys (resp. inputs). For this note that in such a case source S can simply obfuscate the initial circuit (i.e., the circuit that was adapted to introduce weak keys). In any case, the existence of such weaknesses in the hash function to achieve UCE security seem to be a counterintuitive design principle.

Finally, BHK suggest using the SHA family [Nat12] in the HMAC mode [BCK96] as a practical instantiations of UCEs [BHK13a]. It is not known whether any of the SHA functions have weak keys, and indeed, the discovery of such keys would constitute a major breakthrough in the cryptanalysis of SHA. Thus, in order to gain further confidence in the applicability of this construction its extractability properties, for example, in conjunction with the candidate obfuscator of Garg et al. [GGH+13], needs to be better understood.

Acknowledgments

We thank Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi for their personal communication [BHK13d]. Christina Brzuska was supported by the Israel Science Foundation (grant 1076/11 and 1155/11), the Israel Ministry of Science and Technology grant 3-9094), and the German-Israeli Foundation for Scientific Research and Development (grant 1152/2011). Pooya Farshim was supported by grant Fi 940/4-1 of the German Research Foundation (DFG). Arno Mittelbach is supported by CASED (`www.cased.de`).

References

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th Annual Symposium on Foundations of Computer Science*, pages 166–175, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press. (Cited on pages 4 and 11.)

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006. (Cited on pages 4, 11, and 14.)

[AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 166–184, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany. (Cited on page 11.)

[App13] Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. Cryptology ePrint Archive, Report 2013/699, 2013. `http://eprint.iacr.org/2013/699`. (Cited on page 11.)

[BBP04] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 171–188, Interlaken, Switzerland, May 2–6, 2004. Springer, Berlin, Germany. (Cited on page 1.)

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Berlin, Germany. (Cited on page 19.)

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 52–73, San Diego, CA, USA, February 24–26, 2014. Springer, Berlin, Germany. (Cited on page 6.)

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Berlin, Germany. (Cited on pages 3 and 6.)

[BGK$^+$14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks (to appear in EUROCRYPT 2014), 2014. (Cited on page 6.)

[BHK13a] Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Instantiating random oracles via UCEs. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 398–415, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany. (Cited on pages 2, 3, 5, 9, 16, and 19.)

[BHK13b] Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Instantiating random oracles via UCEs. Cryptology ePrint Archive, Report 2013/424., Aug 1, 2013. (Latest version prior to our attack [BHK13d].) http://eprint.iacr.org/2013/424/20130801:043135.). (Cited on pages 2, 3, 6, and 9.)

[BHK13c] Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Instantiating random oracles via UCEs. Cryptology ePrint Archive, Report 2013/424., Oct 17, 2013. (Latest version at the time of writing.) http://eprint.iacr.org/2013/424. (Cited on pages 2, 3, 4, 6, 9, 10, and 16.)

[BHK13d] Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Personal communication. Sep, 2013. (Cited on pages 3, 8, 9, 19, and 21.)

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press. (Cited on page 1.)

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Berlin, Germany. (Cited on pages 5 and 7.)

[BR13] Zvika Brakerski and Guy N. Rothblum. Obfuscating conjunctions. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 416–434, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany. (Cited on page 6.)

[BR14a] Zvika Brakerski and Guy N. Rothblum. Black-box obfuscation for d-cnfs. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, ITCS '14, pages 235–250, New York, NY, USA, 2014. ACM. (Cited on page 6.)

[BR14b] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 1–25, San Diego, CA, USA, February 24–26, 2014. Springer, Berlin, Germany. (Cited on page 6.)

[BRS02] John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 62–75, St. John's, Newfoundland, Canada, August 15–16, 2002. Springer, Berlin, Germany. (Cited on page 9.)

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th Annual ACM Symposium on Theory of Computing*, pages 209–218, Dallas, Texas, USA, May 23–26, 1998. ACM Press. (Cited on page 1.)

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. On the random-oracle methodology as applied to length-restricted signature schemes. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 40–57, Cambridge, MA, USA, February 19–21, 2004. Springer, Berlin, Germany. (Cited on page 1.)

[DOP05] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 449–466, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Berlin, Germany. (Cited on page 1.)

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press. (Cited on pages 3, 6, and 19.)

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 74–94, San Diego, CA, USA, February 24–26, 2014. Springer, Berlin, Germany. (Cited on page 6.)

[GIS$^+$10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 308–326, Zurich, Switzerland, February 9–11, 2010. Springer, Berlin, Germany. (Cited on page 11.)

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th Annual Symposium on Foundations of Computer Science*, pages 102–115, Cambridge, Massachusetts, USA, October 11–14, 2003. IEEE Computer Society Press. (Cited on page 1.)

[GR07] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 194–213, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Berlin, Germany. (Cited on pages 3 and 8.)

[HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation (to appear in EUROCRYPT 2014), 2014. (Cited on page 6.)

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science*, pages 294–304, Redondo Beach, California, USA, November 12–14, 2000. IEEE Computer Society Press. (Cited on page 11.)

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, pages 244–256, 2002. (Cited on page 11.)

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 433–442, Victoria, British Columbia, Canada, May 17–20, 2008. ACM Press. (Cited on pages 4, 11, 12, and 14.)

[KP09] Eike Kiltz and Krzysztof Pietrzak. On the security of padding-based encryption schemes - or - why we cannot prove OAEP secure in the standard model. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 389–406, Cologne, Germany, April 26–30, 2009. Springer, Berlin, Germany. (Cited on page 1.)

[MH14] Takahiro Matsuda and Goichiro Hanaoka. Chosen ciphertext security via UCE. In *PKC 2014: 17th International Workshop on Theory and Practice in Public Key Cryptography*, Lecture Notes in Computer Science. Springer, Berlin, Germany, 2014. (Cited on page 2.)

[Nat12] National Institute of Standards and Technology. FIPS 180-4, Secure Hash Standard (SHS). Technical report, March 2012. (Cited on pages 5 and 19.)

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Berlin, Germany. (Cited on page 1.)

[RSS11] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 487–506, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Germany. (Cited on page 5.)

[SW13] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. Cryptology ePrint Archive, Report 2013/454, 2013. `http://eprint.iacr.org/2013/454`. (Cited on pages 6 and 11.)