Replacing a Random Oracle: Full Domain Hash From Indistinguishability Obfuscation

Susan Hohenberger
Johns Hopkins University
susan@cs.jhu.edu

Amit Sahai
UCLA
sahai@cs.ucla.edu

Brent Waters
University of Texas at Austin
bwaters@cs.utexas.edu

August 16, 2013

Abstract

Our main result gives a way to instantiate the random oracle with a concrete hash function in "full domain hash" applications.

The term full domain hash was first proposed by Bellare and Rogaway [BR93, BR96] and referred to a signature scheme from any trapdoor permutation that was part of their seminal work introducing the random oracle heuristic. Over time the term full domain hash has (informally) encompassed a broader range of notable cryptographic schemes including the Boneh-Franklin [BF01] IBE scheme and Boneh-Lynn-Shacham (BLS) [BLS01] signatures.

All of the above described schemes required a hash function that had to be modeled as a random oracle to prove security. Our work utilizes recent advances in indistinguishability obfuscation to construct specific hash functions for use in these schemes. We then prove security of the *original* cryptosystems when instantiated with our specific hash function.

Of particular interest, our work evades the impossibility result of Dodis, Oliveira, and Pietrzak [DOP05], who showed that there can be no black-box construction of hash functions that allow Full-Domain Hash Signatures to be based on trapdoor permutations. This indicates that our techniques applying indistinguishability obfuscation may be useful in the future for circumventing other such black-box impossibility proofs.

Susan Hohenberger is supported for this research effort by the National Science Foundation (NSF) CNS-1154035 and CNS-1228443; the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-11-2-0211, DARPA N11AP20006, the Office of Naval Research under contract N00014-11-1-0470, and a Microsoft Faculty Fellowship.

Amit Sahai is supported for this research effort in part from NSF grants 1228984, 1136174, 1118096, 1065276, 0916574 and 0830803, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. The views expressed are those of the author and do not reflect the official policy or position of the National Science Foundation, or the U.S. Government.

Brent Waters is upported by NSF CNS-0915361 and CNS-0952692, CNS-1228599 DARPA through the U.S. Office of Naval Research under Contract N00014-11-1-0382, DARPA N11AP20006, Google Faculty Research award, the Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and Packard Foundation Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Defense or the U.S. Government.

1 Introduction

Since the introduction of the Random Oracle Model by Bellare and Rogaway [BR93], a major effort in cryptography has been to understand when and if random oracles can be instantiated with families of actual hash functions while maintaining security. Over the years, we have seen real progress in this effort: Firstly we have seen the discovery of alternative schemes that do not require random oracles but achieve the same security properties as earlier schemes that do require random oracles. For example, Cramer and Shoup [CS98] achieved efficient chosen ciphertext security from DDH hard groups. As another example Canetti, Halevi, and Katz [CHK07] achieved secure IBE without random oracles, following the seminal work of [BF01] giving IBE in the Random Oracle Model. More recently, we have seen the discovery of schemes that not only work in the standard model without random oracles, but work in a manner very similar to the original schemes that used random oracles (e.g. [HSW13, FHPS13] following schemes in the random oracle model [BF01, BLS01]). However, all of these schemes proven secure without random oracles required *changing the underlying cryptographic scheme* in addition to instantiating the random oracle with a concrete hash function. Thus, despite these advances, the following basic question has remained open:

Can we instantiate the random oracle with an actual family of hash functions for existing cryptographic schemes in the random oracle model, such as Full Domain Hash signatures?

In other words, can we achieve security *without changing the underlying cryptographic scheme* at all, but only by replacing the random oracle with a specific family of hash functions? In this work, we give the first positive answer to this question. We do this by leveraging the notion of indistinguishability obfuscation [BGI$^+$01, BGI$^+$12] that was recently achieved in the work of [GGH$^+$13].

Our result is particularly interesting in light of negative results on the Random Oracle Model [CGH98, GK03, BBP04] which have called into question the secure applicability of the Random Oracle Model. Our work is the first to show natural examples of schemes that were originally invented with the Random Oracle Model in mind, that nevertheless remain secure when the random oracle is specifically instantiated.

In particular, our work evades the impossibility result of Dodis, Oliveira, and Pietrzak [DOP05], who showed that there can be no black-box construction of hash functions that allow Full-Domain Hash Signatures to be based on trapdoor permutations. Because we make use of obfuscation, our constructions are inherently non-black-box, and thus are not ruled out by this type of black-box impossibility result. This indicates that our techniques applying indistinguishability obfuscation may be useful in the future for circumventing other such black-box impossibility proofs.

Our Result. Our main result gives a way to instantiate the random oracle with a concrete hash function in "full domain hash" signatures. The full domain hash signature scheme was first proposed[1] in the original Bellare-Rogaway [BR93] paper as a way to build a signature scheme from any trapdoor permutation using the introduced random oracle heuristic. This work was very influential and formed the foundation for part of the PKCS#1 standard [KS98]. While the terminology of "full-domain hash" (FDH) originally applied to the trapdoor permutation signature scheme of Bellare and Rogaway, over time it has (informally) encompassed a broader range of notable cryptographic schemes including the Boneh-Franklin [BF01] IBE scheme, the Cock's IBE scheme [Coc01], and Boneh-Lynn-Shacham (BLS) [BLS01] signatures. Although these schemes exist in different algebraic domains and have different aims, they share common construction and proof structures that uses random oracle programming in very similar ways.

Our work develops a methodology for replacing the programming of a random oracle in these construction using indistinguishable obfuscation in a novel manner. We begin by describing a scheme that replaces the RO hash function in the original Bellare-Rogaway trapdoor permutation (TDP) signature scheme. Our newly instantiated scheme is then proven to be selectively secure.

Let's begin by informally recalling the Bellare-Rogaway TDP-based full domain hash scheme. The signature setup algorithm generates a trapdoor permutation pair of functions $g_{\mathrm{PK}}, g_{\mathrm{SK}}^{-1}$. In addition, it chooses

[1]The terminology "full-domain hash" was actually introduced by Bellare-Rogaway in 1996 [BR96]. They applied this label to the noted signature scheme of their earlier work.

a hash function $H(\cdot)$ that maps from the message space to the domain (and co-domain) of the permutation. The permutation g_{PK} and hash function are published as the verification key and the inverse g_{SK}^{-1} is kept secret. To sign a message m, the signer computes $g_{\mathrm{SK}}^{-1}(H(m))$. To verify a signature σ on message m, the verifier simply checks whether $g_{\mathrm{PK}}(\sigma) \overset{?}{=} H(m)$.

The proof of the Bellare-Rogaway FDH system utilizes the random oracle heuristic to model $H(\cdot)$ as a programmable random oracle. Suppose a poly-time attacker makes at most Q_H oracle queries. One can create a reduction algorithm to the security of the trapdoor permutation as follows. For all but one of the (unique) queries of a message m to the oracle, the reduction algorithm chooses a random value t from the domain and outputs $g_{\mathrm{PK}}(t)$ as the result of the query. For any of these messages it is easy for the reduction algorithm to generate a signature on. It simply outputs t. However, at one query point m^* it programs the output of the random oracle to be $z^* = g_{\mathrm{PK}}(t^*)$ where z^* was given from the trapdoor permutation challenger. If the attacker forges at this message, then the forgery will be t^* which is immediately the solution for the trapdoor permutation inversion.

Our first result is creating a *replacement* hash function for the oracle $H(\cdot)$ and developing a security proof without relying on the random oracle heuristic. To keep with our original goals, our only modifications will be to $H(\cdot)$ and we *will use the signature system construction as is*, with no changes to the underlying trapdoor permutation family. The two main tools we use to build $H(\cdot)$ are an indistinguishability obfuscator [BGI+01, GGH+13] and a recently introduced [BW13, BGI13, KPTZ13] primitive certain called constrained PRFs. In short, a constrained PRF key is a secret key K that allows the evaluator to evaluate the a PRF at a limited set of points, while the rest will appear pseudorandom to him. For our results, we only need a simple form of constrained PRFs called "punctured PRFs" [SW13]. In this setting a private key will be associated with a polynomial set S, where a key $K(S)$ can evaluate the PRF $F(K, x)$ at all x except when $x \in S$. For our proofs we only ever need S to be a singleton set.

We now overview the hash function construction and how we prove it to be selectively secure. (One could use the usual complexity leveraging arguments to claim adaptive security, but we will address adaptive security in a direct way shortly.) To create the hash function the reduction algorithm first chooses a puncturable PRF key K (note this "master key" can evaluate the PRF at all points). Next, the hash function itself will be an obfuscation of the program which on input m computes $g_{\mathrm{PK}}(F(K, m))$. That is the program simply computes the PRF at point m and then applies the trapdoor permutation. We call this program Full Domain Hash. To prove security we will apply the "punctured programs" method of Sahai and Waters [SW13], where we surgically remove a key element of a program, but in a way that does not alter input/output functionality.

Our security proof is formed from a sequence of hybrids. In the first hybrid, we replace the obfuscation of the program Full Domain Hash with an obfuscation of an equivalent program called Full Domain Hash*. This program operates the same as the original except on input m^*, where m^* is the message the attacker selectively chose to attack (before seeing the verification key). At this point instead of computing $F(K, m^*)$ the program is simply hardwired to output a constant z^* to output where z^* is set to be $F(K, m^*)$. Since $z^* = F(K, m^*)$, the input/output behavior is identical. In addition, the program is not given the full PRF key K, but instead is given a punctured PRF key $K(\{m^*\})$. By the security of indistinguishable obfuscation the advantage of any poly-time attacker must be negligibly close between these hybrids. In the next hybrid experiment we replace z^* with a random value chosen from the domain/range of the permutation. The advantage between of this hybrid must also be close due to the constrained PRF security. Now we are finally in a position where we can reduce to the security of the trapdoor permutation. The reduction algorithm receives a TDP challenge z^* and hardcodes that in as the output of $H(m^*)$. It can use a signature on this to invert the challenge. At all other points it knows the punctured PRF key and can therefore compute valid signatures without knowing the inverse of the trapdoor permutation.

We remark that our reduction actually shares some of the spirit of the original random oracle reduction, where a challenge is programmed in at one point and signatures are made by knowing the pre images at all others. A key aspect is that the obfuscation hides the fact that at a certain hybrid m^* is treated differently. If an attacker were able to see inside the obfuscation it could actually see the preimages and break the scheme. Another interesting aspect is that our proof does not leverage the fact that the function $g_{PK}(\cdot)$ is a permutation. It would go through equally well if we only assumed that it was an injective trapdoor function.

Overcoming the black-box impossibility We can now see more precisely why our work evades the impossibility result of Dodis, Oliveira, and Pietrzak [DOP05]. Our hash function is obfuscation of code that runs the underlying permutation. The obfuscation will intuitively hide the evaluation of this code. In particular, no attacker can tell if the trapdoor permutation was actually computed on an input or whether it was a special point where the output was hardcoded in. In the DOP negative result, they build an attack oracle that specifically leverages the black box access to the TDP to watch whenever it is called. It is interesting to see this very strong correlation between the negative result and how non-black box access to a primitive and indistinguishability obfuscation can combine to circumvent it.

Getting Adaptive Security. For our next result we show how to get adaptive (or standard) signature security without complexity leveraging for the case where the trapdoor permutation is the RSA function. The use of RSA as a trapdoor permutation candidate was suggested in Bellare-Rogaway'93 [BR93] and explicitly given in Bellare-Rogaway'96 [BR96]. The public parameters in their scheme are an RSA modulus $N = pq$ for hidden primes p, q and an RSA exponent e chosen such that $\gcd(\phi(N), e) = 1$. The secret key is the integer d where $d \cdot e = 1 \mod \phi(N)$. A signature on message m is of the form $H(m)^d \mod N$ and one verifies a signature σ by checking if $H(m) \overset{?}{=} \sigma^e \mod N$.

We develop a different set of techniques that can leverage the particular structure of the RSA function. The first new ingredient is use of admissible hash functions first introduced in the context of Identity-Based Encryption by Boneh-Boyen [BB04a]. We use a simplification due to Freire et. al. [FHPS13]. At a high level the system is a pair of a hash function $h : \{0,1\}^{\ell(\lambda)} \to \{0,1\}^{n(\lambda)}$ that hashes from the message space to n bit strings and an efficient randomized algorithm AdmSample. The sampling algorithm takes in the security parameter as well as second parameter Q which intuitively corresponds to the number of signature queries an attacker makes. It outputs a string $u \in \{0, 1, \perp\}^n$. Informally, we say that the system is admissible if the following conditions hold. Consider any sequence of Q values $x_1, \ldots, x_Q$ and $x^* \neq x_i$. The event we consider is where the string $h(x_i)$ has a bit in common with u in at least one position, but $h(x^*)$ is different from u at all positions. (Note, if $u_j = \perp$ then it is different at position j from all bit strings.) If this event occurs with non-negligible probability, we say it is an admissible system. Intuitively, when used in a proof of a signature scheme, the admissible hash function is utilized to partition the message space into messages that can be signed in the query phase and those that can be used in the challenge phase. A sampled string u corresponds to a particular partition. When running a reduction, one hopes that the actual signature oracle queries and forgery message align with a partition, and the reduction aborts otherwise.

To build the hash function candidate, the setup first chooses a random $v \in \mathbb{Z}_N^*$ as well as exponents $a_{i,b}$ chosen randomly in $[0, \phi(N)]$, for all $i \in [1, n], b \in \{0, 1\}$. Next, it builds the hash function as an obfuscation of the program RSA Hash. The program will first compute $m' = h(m)$. Then, it computes and outputs $v^{\prod_{i \in [n]} a_{i,m'_i}}$.

Our proof proceeds in a few hybrid steps. In the first hybrid experiment the challenger creates a partition internally by calling AdmSample$(1^\lambda, Q) \to u$ for an attacker that makes at most $Q = Q(\lambda)$ queries. The game aborts and declares the attacker unsuccessful if any of the query messages or forgery message violates the partition. The property of admissible hashes states any attacker with non-negligible advantage in the real game will also have non-negligible advantage here. In the next hybrid, we change the way we sample the exponents $a_{i,b}$. One first chooses random $y_{i,b} \in [1, N]$. Then for when $u_i = b$ we set $c_{i,b} = e \cdot y_{i,b}$. If $u_i \neq b$ we set $c_{i,b} = e \cdot y_{i,b} + 1$. Note in the first case $c_{i,b}$ is a multiple of e and in the second case $e \nmid c_{i,b}$. The values $a_{i,b} = c_{i,b} \mod \phi(N)$. We show that this way of choosing a values is statistically close to the previous uniform way, because $\gcd(\phi(N), e) = 1$.

Next, we use an alternative program where we directly use the $c_{i,b}$ values in place of the $a_{i,b}$ values. Since the group $\mathbb{Z}_N^*$ is of order $\phi(N)$ we have that $v^{\prod_{i \in [n]} a_{i,m'_i}} = v^{\prod_{i \in [n]} c_{i,m'_i}}$ for all m'. Therefore the input/output behavior is the same between the two programs and we can argue the advantage in the hybrids for poly-time attackers must be close by indistinguishability obfuscation. This is the critical hybrid experiment in that it most radically departs from previous such proofs, by leveraging indistinguishability obfuscation. Observe that this hybrid experiment eliminates the need for the reduction to know $\phi(N)$, which is crucial to the reduction, since it uses $c_{i,b}$ values instead of $a_{i,b}$ values. However, if the values $c_{i,b}$ were completely visible to an attacker, they would be trivially distinguishable from the "true" uniform $a_{i,b}$ values. However, indistinguishability obfuscation guarantees that these values are hidden from the attacker, and that indeed the attacker cannot distinguish this hybrid from the previous one.

Finally, we show that any attacker that is successful in the last hybrid can be used to break the RSA assumption. For any signature query message m that respects the partition, the reduction will view $H(m)$ as v raised to some integer that is a multiple of e and taking the e-th root is then easy. Any forgery on m^* that respects the partition, the reduction will view $H(m^*)$ as v^z for some z where $\gcd(e, z) = 1$ and from this can derive $v^{1/e}$.

BLS Signatures and More We extend our techniques to replacing the random oracle in the BLS [BLS01] signature scheme. In Section 5 we give a candidate that has a selective proof of security based on the computational Diffie-Hellman problem (along with indistinguishability obfuscation). In Section 6, we give an adaptive proof of security based on an assumption equivalent to the n-Diifie-Hellman inversion assumption. The high level structures of these are similar to the respective selective and adaptive construction and proof methods above. The lower level mechanisms are adapted to the context of bilinear groups. Finally, in Section 7 we sketch how the BLS ideas extend to the Boneh-Franklin IBE scheme.

1.1 Other Related Work

Recently, the work of [BHK13] looked at a complementary question of identifying a definitional abstraction to replace the random oracle heuristic in many random oracle-based constructions. The abstraction is a notion of security called UCE (Universal Computational Extractor). The authors emphasize that a random oracle is *known* not to exist and "behaves like a random oracle" is not a rigorously defined property, whereas UCE is a well defined property of a hash function. They then show how several previous constructions proven secure in the random schemes can be proven secure if we assume the hash functions are UCE secure. One can then conjecture that standard cryptographic hash functions like SHA-256 may satisfy the UCE security notion. In contrast, our work is focused on providing *new* candidate constructions for hash functions, that allow for a security proof to work with the original constructions in the random oracle model. Interestingly, the work of [BHK13] does not encompass the case of Full Domain Hash signatures, arguably one of the most natural and well-studied constructions in the Random Oracle Model, that we address here.

2 Preliminaries

In this section, we define indistinguishability obfuscation, and variants of pseudo-random functions (PRFs) that we will make use of. All the variants of PRFs that we consider will be constructed from one-way functions.

2.1 Indistinguishability Obfuscation

The definition below is from [GGH$^+$13]; there it is called a "family-indistinguishable obfuscator", however they show that this notion follows immediately from their standard definition of indistinguishability obfuscator using a non-uniform argument.

Definition 1 (Indistinguishability Obfuscator ($i\mathcal{O}$))**.** A uniform PPT machine $i\mathcal{O}$ is called an *indistinguishability obfuscator* for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:

- For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x, we have that

$$\Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(\lambda, C)] = 1$$

- For any (not necessarily uniform) PPT adversaries $Samp$, D, there exists a negligible function α such that the following holds: if $\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \tau) \leftarrow Samp(1^\lambda)] > 1 - \alpha(\lambda)$, then we have:

$$\Big| \Pr\left[D(\tau, i\mathcal{O}(\lambda, C_0)) = 1 : (C_0, C_1, \tau) \leftarrow Samp(1^\lambda)\right]$$
$$- \Pr\left[D(\tau, i\mathcal{O}(\lambda, C_1)) = 1 : (C_0, C_1, \tau) \leftarrow Samp(1^\lambda)\right] \Big| \leq \alpha(\lambda)$$

In this paper, we will make use of such indistinguishability obfuscators for all polynomial-size circuits:

Definition 2 (Indistinguishability Obfuscator for $P/poly$)**.** A uniform PPT machine $i\mathcal{O}$ is called an *indistinguishability obfuscator* for $P/poly$ if the following holds: Let $\mathcal{C}_\lambda$ be the class of circuits of size at most λ. Then $i\mathcal{O}$ is an indistinguishability obfuscator for the class $\{\mathcal{C}_\lambda\}$.

Such indistinguishability obfuscators for all polynomial-size circuits were constructed under novel algebraic hardness assumptions in [GGH$^+$13].

2.2 Constrained PRFs

We first consider some simple types of constrained PRFs [BW13, BGI13, KPTZ13], where a PRF is only defined on a subset of the usual input space. We focus on *puncturable* PRFs, which are PRFs that can be defined on all bit strings of a certain length, except for any polynomial-size set of inputs:

Definition 3. A *puncturable* family of PRFs F mapping is given by a triple of Turing Machines Key_F, $\mathrm{Puncture}_F$, and Eval_F, and a pair of computable functions $n(\cdot)$ and $m(\cdot)$, satisfying the following conditions:

- **[Functionality preserved under puncturing]** For every PPT adversary A such that $A(1^\lambda)$ outputs a set $S \subseteq \{0,1\}^{n(\lambda)}$, then for all $x \in \{0,1\}^{n(\lambda)}$ where $x \notin S$, we have that:

$$\Pr\left[\mathsf{Eval}_F(K, x) = \mathsf{Eval}_F(K_S, x) : K \leftarrow \mathrm{Key}_F(1^\lambda), K_S = \mathrm{Puncture}_F(K, S)\right] = 1$$

- **[Pseudorandom at punctured points]** For every PPT adversary (A_1, A_2) such that $A_1(1^\lambda)$ outputs a set $S \subseteq \{0,1\}^{n(\lambda)}$ and state τ, consider an experiment where $K \leftarrow \mathrm{Key}_F(1^\lambda)$ and $K_S = \mathrm{Puncture}_F(K, S)$. Then we have

$$\Big| \Pr\left[A_2(\tau, K_S, S, \mathsf{Eval}_F(K, S)) = 1\right] - \Pr\left[A_2(\tau, K_S, S, U_{m(\lambda)\cdot|S|}) = 1\right] \Big| = negl(\lambda)$$

where $\mathsf{Eval}_F(K, S)$ denotes the concatenation of $\mathsf{Eval}_F(K, x_1)), \ldots, \mathsf{Eval}_F(K, x_k))$ where $S = \{x_1, \ldots, x_k\}$ is the enumeration of the elements of S in lexicographic order, $negl(\cdot)$ is a negligible function, and U_ℓ denotes the uniform distribution over ℓ bits.

For ease of notation, we write $F(K, x)$ to represent $\mathsf{Eval}_F(K, x)$. We also represent the punctured key $\mathrm{Puncture}_F(K, S)$ by $K(S)$.

The GGM tree-based construction of PRFs [GGM84] from one-way functions are easily seen to yield puncturable PRFs, as recently observed by [BW13, BGI13, KPTZ13]. Thus we have:

Theorem 1. [GGM84, BW13, BGI13, KPTZ13] If one-way functions exist, then for all efficiently computable functions $n(\lambda)$ and $m(\lambda)$, there exists a puncturable PRF family that maps $n(\lambda)$ bits to $m(\lambda)$ bits.

Next we consider families of PRFs that are with high probability injective:

2.3 RSA Assumption and Shamir's Lemma

We begin by recalling (one of the) standard versions of the RSA assumption [RSA78].

Assumption 1 (RSA)**.** Let λ be the security parameter. Let positive integer N be the product of two λ-bit, distinct odd primes p, q. Let e be a randomly chosen positive integer less than and relatively prime to $\phi(N) = (p-1)(q-1)$. Given (N, e) and a random $y \in \mathbb{Z}_N^*$, it is hard to compute x such that $x^e \equiv y \mod N$.

We also make use of the following lemma due to Shamir.

Lemma 1 (Shamir [Sha83])**.** Given $x, y \in \mathbb{Z}_N$ together with $a, b \in \mathbb{Z}$ such that $x^a = y^b \pmod{N}$ and $\gcd(a, b) = 1$, there is an efficient algorithm for computing $z \in \mathbb{Z}_N$ such that $z^a = y \pmod{N}$.

2.4 Bilinear Groups and the CDH Assumption

Let $\mathbb{G}$ and $\mathbb{G}_T$ be groups of prime order p. A *bilinear map* is an efficient mapping $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ which is both: (*bilinear*) for all $g \in \mathbb{G}$ and $a, b \leftarrow \mathbb{Z}_p$, $e(g^a, g^b) = e(g, g)^{ab}$; and (*non-degenerate*) if g generates $\mathbb{G}$, then $e(g, g) \neq 1$.

Assumption 2 (Computational Diffie-Hellman)**.** Let g generate a group $\mathbb{G}$ of prime order $p \in \Theta(2^\lambda)$. For all p.p.t. adversaries $\mathcal{A}$, the following probability is negligible in λ:

$$\Pr[a, b \leftarrow \mathbb{Z}_p; z \leftarrow \mathcal{A}(g, g^a, g^b) : z = g^{ab}].$$

2.5 The n-Diffie-Hellman Inversion Assumption and Equivalent Formulation

Our Section 6 construction of adaptively secure BLS signatures makes use of the n-Diffie-Hellman Inversion assumption [BB04b]. This is a parameterized family of assumptions, where the number of group elements involved increases with n.

Assumption 3 (n-Diffie-Hellman Inversion)**.** Let h generate a group $\mathbb{G}$ of prime order $p \in \Theta(2^\lambda)$. For all p.p.t. adversaries $\mathcal{A}$, the following probability is negligible in λ:

$$\Pr[b \leftarrow \mathbb{Z}_p; z \leftarrow \mathcal{A}(h, h^b, h^{b^2}, \ldots, h^{b^n}) : z = g^{1/b}].$$

We will actually use an equivalent assumption, which is easier to work with in our proof. We call the assumption the n-DHI Equivalent assumption for the purposes of this paper. The assumption is stated as follows.

Assumption 4 (n-DHI Equivalent)**.** Let g generate a group $\mathbb{G}$ of prime order $p \in \Theta(2^\lambda)$. For all p.p.t. adversaries $\mathcal{A}$, the following probability is negligible in λ:

$$\Pr[a \leftarrow \mathbb{Z}_p; z \leftarrow \mathcal{A}(g, g^a, g^{a^2}, \ldots, g^{a^n}) : z = g^{a^{n+1}}].$$

We briefly sketch why an attacker $\mathcal{A}$ on the new assumption implies an attacker on the n-DHI assumption. Suppose, that an algorithm $\mathcal{B}$ is given a DHI instance $h, h^b, h^{b^2}, \ldots, h^{b^n}$. Then, it creates an instance for $\mathcal{A}$ by setting $g = h^{b^n}, g^a = g^{b^{n-1}}, \ldots, g^{a^n} = h$. Essentially, this sets $g = h^{b^n}$ and implicitly $a = b^{-1}$. Therefore, $g^{a^{n+1}} = h^{1/b}$. Thus an efficient attacker to the n-DHI Equivalent problem implies one to the n-DHI problem. The other direction of equivalence follows in an analogous manner.

3 Full-Domain Hash Signatures (Selectively Secure)

In this section, we revisit the Bellare-Rogaway Full-Domain Hash (FDH) signature scheme [BR93, BR96], and show how to make it selectively secure in the standard model by instantiating the random oracle in a specific way. We stress that we do not modify the Bellare-Rogaway FDH signature scheme in any way; the only new aspect of our construction is our instantiation of the random oracle with a specific function whose description becomes part of the public key.

Recall that the Bellare-Rogaway FDH signature scheme required a trapdoor permutation family. Our method, in fact, not only applies to trapdoor permutation families, but indeed to any *injective* trapdoor function family. We prove the selective security of the FDH signature scheme based on the security of the indistinguishability obfusctor, the security of a puncturable PRF family, and the security of an injective trapdoor function family.

For simplicity of exposition, we assume that there is a polynomial $\ell(\lambda)$ which denotes the length of messages to be signed; we denote this message space by $\mathcal{M} = \{0,1\}^{\ell(\lambda)}$. More generally, a collision-resistant hash function may be used to hash messages to this size.

- $\mathsf{Setup}(1^\lambda)$: The setup algorithm first runs $\mathsf{TDFSetup}(1^\lambda)$ and that produces a public index PK along with a trapdoor SK, yielding the map $g_{\mathrm{PK}} : \{0,1\}^n \to \{0,1\}^w$ together with its inverse. Next, the setup algorithm chooses a puncturable PRF key K for F where $F(K,\cdot) : \{0,1\}^{\ell(\lambda)} \to \{0,1\}^n$. Then, it creates an obfuscation of the of the program Full Domain Hash Figure 1. The size of the program is padded to be the maximum of itself and the program Full Domain Hash* of Figure 2. We refer to the obfuscated program as the function $H : \{0,1\}^{\ell(\lambda)} \to \{0,1\}^w$, which acts as the random oracle type hash function in the Bellare-Rogaway scheme.

  The verification key VK consists of the trapdoor index PK as well as the hash function $H(\cdot)$. The secret key is the trapdoor SK as well as $H(\cdot)$.

- $\mathsf{Sign}(\mathrm{SK}, m \in \mathcal{M})$: The signature algorithm outputs $\sigma = g^{-1}_{\mathrm{SK}}(H(m)) \in \{0,1\}^n$.

- $\mathsf{Verify}(\mathrm{VK}, m, \sigma)$ The verification algorithm tests if $g_{\mathrm{PK}}(\sigma) \overset{?}{=} H(m)$ and outputs accept if and only if this holds.

Full Domain Hash

Constants: PRF key K, trapdoor function index PK.
Input: Message m.

1. Output $g_{\mathrm{PK}}(F(K,m))$.

Figure 1: Full Domain Hash

Full Domain Hash*

Constants: Punctured PRF key $K(\{m^*\})$, $m^* \in \mathcal{M}$, $z^* \in \{0,1\}^w$, trapdoor function index PK.
Input: Message m.

1. If $m = m^*$ output z^* and exit.
2. Else output $g_{\mathrm{PK}}(F(K,m))$.

Figure 2: Full Domain Hash*

Theorem 2. If our obfuscation scheme is indistingishuably secure, F is a secure punctured PRF, and the injective trapdoor function is secure, then the above signature scheme is selectively secure.

We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original signature security game. We prove that a poly-time attacker's advantage must be negligibly close between each successive one. Then, we show that any poly-time attacker in the final experiment that succeeds in forging with non-negligible probability can be used to invert the injective trapdoor function.

- Hyb_0 : In the first hybrid the following game is played:
  1. The attacker selectively gives the challenger the message m^*.
  2. The TDF index is chosen by the challenger running $\mathsf{TDFSetup}(1^\lambda)$.
  3. K is chosen as a key for the puncturable PRF.
  4. The hash function $H(\cdot)$ is created as an obfuscation of the program Full Domain Hash.
  5. The attacker queries the sign oracle a polynomial number of times on messages $m \neq m^*$. It receives back $g^{-1}_{\mathrm{SK}}(H(m)) = F(K, m)$. (Note the equality holds since the function g_{PK} is injective.)
  6. The attacker sends a forgery σ^* and wins if $\mathsf{Verify}(\mathrm{VK}, m^*, \sigma^*) = 1$.
- Hyb_1 : Is the same as Hyb_0 except we let $z^* = g_{\mathrm{PK}}(F(K, m^*))$ and let VK be the obfuscation of the program Verify Signature* of Figure 2.
- Hyb_2 : Is the same as Hyb_1 except $z^* = g_{\mathrm{PK}}(t)$ for t chosen uniformly at random in $\{0,1\}^n$.

Lemma 2. If our obfuscation scheme is indistinguishability secure, then the advantage of a poly-time attacker in Hyb_0 is negligibly close to the advantage in Hyb_1.

Proof. We prove this lemma by giving a reduction to the indistinguishability security of the obfuscator. To do so, we must build the two algorithms *Samp* and D.

Samp(1^λ) behaves as follows: It invokes the adversary to obtain m^* and the adversary's state τ'. It runs $\mathsf{TDFSetup}(1^\lambda)$ to obtain PK and SK. It then chooses K as a key for the puncturable PRF. It sets $z^* = g_{\mathrm{PK}}(F(K, m^*))$. It sets $\tau = (m^*, \mathrm{PK}, \mathrm{SK}, K, \tau')$ and builds C_1 as the program for Full Domain Hash, and C_2 as the program for Full Domain Hash*.

Before describing D, we observe that by construction and the functionality preservation property of puncturable PRFs, the circuits C_1 and C_2 always behave identically on every input. Because of padding, both C_1 and C_2 have the same size. Thus, *Samp* satisfies the conditions needed for invoking the indistinguishability property of the obfuscator.

Now, we can describe the algorithm D, which takes as input τ as given above, and either the obfuscation of C_1, which is the program Full Domain Hash, or C_2, which is the program Full Domain Hash*. D creates the verification key for the signature scheme by combining PK with the obfuscated program as the hash function description. It then invokes the adversary on this verification key, and the adversary then makes requests for signatures on messages $m \neq m^*$. For each such message, D constructs the signatures $g^{-1}_{\mathrm{SK}}(H(m)) = F(K, m)$, through its knowledge of K within τ. Finally, the attacker sends a forgery σ^* and wins if $\mathsf{Verify}(m^*, \sigma^*) = 1$. If the attacker wins, D outputs 1.

By construction, if D receives an obfuscation of C_1, then the probability that D outputs 1 is exactly the probability of the adversary winning in hybrid Hyb_0. On the other hand, if D receives an obfuscation of C_2, then the probability that D outputs 1 is the probability of the adversary winning in hybrid Hyb_1.

The lemma follows. ■

Lemma 3. If our confined PRF is secure, then the advantage of a poly-time attacker in Hyb_1 is negligibly close to the advantage in Hyb_2.

Proof. We prove this lemma by giving a reduction to the pseudorandomness property at punctured points for punctured PRFs. To do so, we must build the algorithms A_1 and A_2.

$A_1(1^\lambda)$ simply invokes the adversary to obtain the challenge message m^* and state τ', and outputs the singleton set $S = \{m^*\}$ and $\tau = (1^\lambda, \tau')$.

A_2 obtains as input τ, the punctured key K_S, the singleton set $S = \{m^*\}$, and either a value $t^* = F(K, m^*)$ or a uniformly random value t^*. Then, A_2 invokes $\mathsf{TDFSetup}(1^\lambda)$ to obtain PK and SK. Now given t^*, it can compute $z^* = g_{\mathrm{PK}}(t^*)$. Note that this yields either the z^* value computed in hybrid Hyb_1 or in hybrid Hyb_2. Since it knows K_S, now A_2 can obfuscate the program Full Domain Hash*, and then execute the adversary and answer its signature queries using the punctured key K_S. Finally, A_2 outputs 1 if the adversary succeeds.

By construction, the pseudorandomness property for punctured PRFs implies the lemma. ■

Lemma 4. If our injective trapdoor function is hard to invert, then the advantage of a poly-time attacker in Hyb_2 is negligible.

Proof. We prove this lemma by giving a reduction to the one-wayness of the injective trapdoor function. To do so, we build an inverting algorithm Inv.

Inv takes as input a public index PK for an injective trapdoor function, and a target $z^* = g_{\mathrm{PK}}(t^*)$ for some (as yet unknown) random value t^*. The algorithm Inv then invokes the adversary to obtain m^*, and chooses a PRF key K and builds the punctured key $K(S)$ where $S = \{m^*\}$. It uses this key, together with PK and z^*, to obfuscate the program Full Domain Hash*. It can then execute the adversary, and use its knowledge of $K(S)$ to answer all adversary signing queries. The adversary then terminates with an attempted forgery σ^* on message m^*. By the definition of the program Full Domain Hash*, this forgery can only be valid if $g_{\mathrm{PK}}(\sigma^*) = z^*$, and because g_{PK} is injective, this can only happen if $\sigma^* = t^*$. Thus if the adversary is successful, Inv can output σ^* as a valid pre-image of z^*.

We observe that by construction of Inv, the probability of success of Inv is exactly the probability that the attacker succeeds in hybrid Hyb_2. The lemma follows. ■

These three lemmas together yield our main theorem that the above full domain hash signature scheme is selectively secure.

4 Adaptively Secure RSA Full Domain Hash Signatures

We first describe at a high level what advantage indistinguishability obfuscation gives us in this situation: In several previous constructions of adaptively secure schemes in the plain model starting with the adaptively secure IBE scheme of [BB04a], a special hash function was chosen that allowed for a "partitioning" proof of security. In essence, for this to work, the hash function should have two "modes":

- In the "normal" mode, the hash function's parameters are typically just chosen at random, and it behaves like an ordinary hash function.
- In the "partitioning" mode, the hash function parameters are chosen according to a special distribution. This special distribution allows for the efficient computation of the inverse of the hash value for a large fraction of points, but it has the property that computing the inverse of the hash value at any other point is computationally hard.

It is crucial that the input/output functionality of the hash function should be identical in the two modes, and we will also use this property. However, in previous proofs (like [BB04a]), it was also critical that the hash function parameters in "partitioning" mode be information theoretically indistinguishable from the parameters in "normal" mode, and thus the partition should be hidden from the adversary even when given the hash function parameters. This restriction significantly limited the applicability of this technique, as it could only be applied with algebraic structures that allowed for such "pseudorandom" hash parameters. Thanks to indistinguishability obfuscation, however, we can avoid this restriction by obfuscating the hash function description. Thus, even if the natural hash function parameters in "partitioning" mode clearly reveal the partition and thus are distinguishable from normal parameters, because the resulting hash function is *functionally* identical to a hash function in "normal" mode, the obfuscated hash function *must* hide the partition, and this allows the proof of adaptive security to go through.

In describing our signature scheme, For simplicity of exposition, we assume that there is a polynomial $\ell(\lambda)$ which denotes the length of messages to be signed; we denote this message space by $\mathcal{M} = \{0,1\}^{\ell(\lambda)}$.

More generally, a collision-resistant hash function may be used to hash messages to this size. Below, for any polynomial in λ, after the first mention of this polynomial, we will often suppress the dependence on λ for ease of notation. Thus, below often we will simply refer to the size of messages to be signed by ℓ.

Before describing our construction, we first recall a (simplified) description of the notion of *admissible* hash functions due to [BB04a]. Our definition is a slight variation of the simplified definition due to [FHPS13].

Definition 4. Let ℓ, n and θ be efficiently computable univariate polynomials. We say that an efficiently computable function $h : \{0,1\}^{\ell(\lambda)} \to \{0,1\}^{n(\lambda)}$, and an efficient randomized algorithm AdmSample, is θ-*admissible* if the following condition holds:

For any $u \in (\{0,1\} \cup \{\perp\})^n$, define $P_u : \{0,1\}^\ell \to \{0,1\}$ as follows: $P_u(x) = 0$ iff $\forall i : h(x)_i \neq u_i$, and otherwise (if $\exists i : h(x)_i = u_i$) we have $P_u(x) = 1$.

Then we require that for any efficiently computable polynomial $Q(\lambda)$, for all $x_1, \ldots, x_Q, z \in \{0,1\}^\ell$, where $z \notin \{x_i\}$, we have that

$$\Pr\left[P_u(x_1) = P_u(x_2) = \cdots = P_u(x_Q) = 1 \ \wedge \ P_u(z) = 0\right] \geq 1/\theta(Q)$$

where the probability is taken only over $u \leftarrow \mathrm{AdmSample}(1^\lambda, Q)$.

Theorem 3 (Admissible Function Families [BB04a], see also [FHPS13] for a simple proof)**.** For any efficiently computable polynomials ℓ, n, there exists an efficiently computable polynomial θ such that there exist $\theta-$admissible function families mapping ℓ bits to n bits.

We now show that we can leverage the structure of the RSA trapdoor permutation to prove adaptive security. The use of RSA as a candidate for a trapdoor permutation was first discussed in the original Bellare-Rogaway [BR93] paper, however, it was in [BR96] that Bellare and Rogaway gave an explicit full domain hash RSA construction. This construction formed the basis for part of the standard PKCS#1 [KS98].

- $\mathsf{Setup}(1^\lambda)$: The setup algorithm first runs an RSA type setup. It chooses random primes p, q of λ bits each. We define $N = p \cdot q$ and $\phi(N) = (p-1)(q-1)$. We let e be a random chosen integer between 1 and $\phi(N)$ such that $\gcd(\phi(N), e) = 1$.

  Next, it chooses integers $(a_{1,0}, a_{1,1}), \ldots, (a_{n,0}, a_{n,1})$ each uniformly at random from the range $[1, \phi(N)-1]$. In addition, it chooses a group element $v \in \mathbb{Z}_N^*$. It then creates an obfuscation of the of the program RSA Hash of Figure 3. The size of the program is padded to be the maximum of itself and the program RSA Hash* of Figure 4. We refer to the obfuscated program as the function $H(\cdot)$. This function $H(\cdot)$ will replace the random oracle in the RSA full domain hash scheme, but no other part of the scheme is modified.

  The verification key VK consists of the integers N, e as well as the hash function $H : \{0,1\}^{\ell(\lambda)} \to \mathbb{Z}_N^*$. The secret key is the integer d where $e \cdot d \equiv 1 \mod \phi(N)$.

- $\mathsf{Sign}(\mathrm{SK}, m \in \mathcal{M})$: The signature algorithm outputs $\sigma = H(M)^d \mod N$.

- $\mathsf{Verify}(\mathrm{VK}, m, \sigma)$ The verification algorithm tests if $\sigma^e \equiv H(m) \mod N$ and outputs accept if and only if this holds.

Remark 1. For simplicity of exposition we describe computing the programs output by first computing a integer $\pi(m')$ as a product of n integers and then raising v to this mod N. In practice, it might be more efficient to incrementally raise an accumulated value to each a_{i,m'_i}.

Theorem 4. If our obfuscation scheme is indistingishuably secure and the RSA assumption holds, the above signature scheme is existentially unforgeable against chosen message attacks.

We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original signature security game. In the first hybrid step we do a "partitioning" of the message space. Consider a poly-time attacker that makes $Q = Q(\lambda)$ signature queries $m_1, \ldots, m_Q$ and attempts to forge on message

RSA Hash

Constants: RSA modulus N, integers $(a_{1,0}, a_{1,1}), \ldots, (a_{n,0}, a_{n,1})$ each in $[1, \phi(N) - 1]$, and $v \in \mathbb{Z}_N^*$.
Input: Message m.

1. Compute $m' = h(m)$.
2. Compute the integer $\pi(m') = \prod_{i \in [n]} a_{i,m'_i}$.
3. Output $v^{\pi(m')} \pmod N$.

Figure 3: RSA Hash

RSA Hash*

Constants: RSA modulus N, integers $(c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1})$ each chosen as in Hyb_2, and $v \in \mathbb{Z}_N^*$.
Input: Message m.

1. Compute $m' = h(m)$.
2. Compute the integer $\pi(m') = \prod_{i \in [n]} c_{i,m'_i}$.
3. Output $v^{\pi(m')} \pmod N$.

Figure 4: RSA Hash*

$m^* \neq m_i$ for all i. Roughly, at the beginning of Hyb_1 the challenger will now (behind the scenes) partition the message space such that a large fraction of messages will fall into a "query" space and a much smaller, but still non-negligible fraction of messages will fall into the "challenge" space. Furthermore, in this new game the attacker is only considered to have won if he both forged a signature *and* all his signature queries $m_1, \ldots, m_n$ fall into the query space and m^* falls into the challenge space. We can show that if an attacker succeeds in the original security game (that does not have these additional restrictions on winning) with non-negligible advantage, then if will succeed in Hyb_1 with non-negligible advantage. Our system uses the Boneh-Boyen [BB04a] admissible hash function defined above, where if an attacker has advantage ϵ in Hyb_0, he will have advantage $\epsilon/\theta(Q)$ in Hyb_1.

After the first proof step we prove that a poly-time attacker's advantage must be negligibly close between each successive hybrid experiment. We finally show that any poly-time attacker in the final experiment that succeeds with non-negligible probability can be used to break the RSA assumption.

- Hyb_0 : In the first hybrid the following game is played.
  1. The challenger sets $N = p \cdot q$ and $\phi(N) = (p-1)(q-1)$. It chooses e as a random chosen integer between 1 and $\phi(N)$ such that $\gcd(\phi(N), e) = 1$.
  2. The challenger chooses integers $(a_{1,0}, a_{1,1}), \ldots, (a_{n,0}, a_{n,1})$ each uniformly at random from the range $[1, \phi(N) - 1]$.
  3. The variable v is chosen uniformly at random in $\mathbb{Z}_N^*$.
  4. The hash function $H(\cdot)$ is created as an obfuscation of the program RSA Hash.
  5. The attacker queries the signing oracle at most Q times on messages $m_1, \ldots, m_Q$. In its ith query, it receives back $H(m_i)^d \pmod N$.
  6. The attacker finally chooses a message m^*, sends a forgery σ^*, and wins if $\mathsf{Verify}(\mathrm{VK}, m^*, \sigma^*) = 1$.
- Hyb_1 : Is the same as Hyb_0 except the challenger begins by sampling a string $u \in (\{0, 1, \perp\})^n$ by calling $\mathrm{AdmSample}(1^\lambda, Q) \to u$, where Q is an upper bound on the number of queries made by the adversary (this could, for example, be the running time of the adversary). At the end of the experiment, the attacker is only considered to be successful if both $\mathsf{Verify}(\mathrm{VK}, m^*, \sigma^*) = 1$ and $P_u(m^*) = 0$ and for all messages m queried, $P_u(m) = 1$. If the attacker is successful but this condition is not satisfied, we say that the hybrid "aborts."

- Hyb_2 : Is the same as Hyb_1 except the for the following modification. After sampling u, the challenger chooses integers $(c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1})$ in the following way. For $i \in [n]$ and $b \in \{0,1\}$, $y_{i,b}$ is chosen uniformly at random from all integers in $[1, N]$. The challenger then sets

$$c_{i,b} = \begin{cases} e \cdot y_{i,b} & \text{if } b = u_i \\ e \cdot y_{i,b} + 1 & \text{if } b \neq u_i \end{cases}$$

  Observe that $\gcd(e, e \cdot y_{i,b} + 1) = 1$, for all i, b. We note that if $b \neq u_i$, then either $u_i = 1 - b$ or $u_i = \perp$. Then for $i \in [n]$ and $b \in \{0,1\}$, it sets $a_{i,b} = (c_{i,b} \bmod \phi(N))$.

- Hyb_3 : Is the same as Hyb_2 except the challenger creates the hash function $H(\cdot)$ as an obfuscation of the program RSA Hash* using the values $(c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1})$. The "$a$" values are no longer computed or used.

Lemma 5. Consider a attacker that makes at most a polynomial of queries $Q = Q(\lambda)$ in Hyb_0. If the advantage of an attacker in Hyb_0 is $\epsilon(\lambda)$, then the advantage of the attacker in Hyb_1 will be at least $\epsilon(\lambda)/\theta(Q)$. In particular, any poly-time attacker with non negligible advantage in Hyb_0 will also have non-negligible advantage in Hyb_1.

Proof. The lemma follows immediately from the function h satisfying the definition of a θ-admissibility, since the only the independent choice of $u \leftarrow \text{AdmSample}(1^\lambda, Q)$ determines whether or not the hybrid aborts. ∎

Lemma 6. The advantage of any attacker in Hyb_1 is negligibly close to the advantage in Hyb_2.

Proof. Fix i, b, and consider the $a_{i,b}$ value that results from the choice of $c_{i,b}$ in hybrid Hyb_2. First, observe that the random choice of $y_{i,b}$ from the range $[1, N]$ is negligibly statistically close to a random choice of $y_{i,b}$ in the range $[1, \phi(N) - 1]$, since $N - \phi(N) = p + q - 1$. Thus, for the remainder of the argument, we can assume that each $y_{i,b}$ is chosen uniformly from the range $[1, \phi(N) - 1]$. Next, since $\gcd(e, \phi(N)) = 1$, we have that both $(e \cdot y_{i,b} \bmod \phi(N))$ and $(e \cdot y_{i,b} + 1 \bmod \phi(N))$ are distributed uniformly in the range $[1, \phi(N) - 1]$, and the lemma follows. ∎

Lemma 7. If our obfuscation scheme is indistinguishability secure, then the advantage of any poly-time algorithm in Hyb_2 is negligibly close to the advantage in Hyb_3.

Proof. We prove this lemma by giving a reduction to the indistinguishability security of the obfuscator. To do so, we must build the two algorithms *Samp* and D.

Samp(1^λ) behaves as follows: It invokes the adversary to obtain the adversary's state τ'. It runs the RSA type setup as in the real scheme to generate primes p, q and sets $N = p \cdot q$ and $\phi(N) = (p-1)(q-1)$. It chooses e as a random integer between 1 and $\phi(N)$ such that $\gcd(\phi(N), e) = 1$. It sets integer d where $e \cdot d \equiv 1 \mod \phi(N)$. It chooses $v \in \mathbb{Z}_N^*$ as a random element. It chooses $(c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1})$ and $(a_{1,0}, a_{1,1}), \ldots, (a_{n,1}, a_{n,0})$ derived from them, as in Hyb_2. It sets $\tau = (N, p, q, e, d, (c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1}), (a_{1,0}, a_{1,1}), \ldots, (a_{n,1}, a_{n,0}), v, \tau')$ and builds C_1 as the program for RSA Hash, and C_2 as the program for RSA Hash*.

Before describing D, we observe that by construction, the circuits C_1 and C_2 always behave identically on every input. To show program equivalence, note that since $\mathbb{Z}_N^*$ is of order $\phi(N)$ for all m', we have that

$$\begin{aligned} v^{\prod_i c_{i,m'_i}} \pmod N &= v^{(\prod_i c_{i,m'_i}) \pmod{\phi(N)}} \pmod N = \\ v^{\prod_i (c_{i,m'_i} \pmod{\phi(N)})} \pmod N &= v^{\prod_i a_{i,m'_i}} \pmod N. \end{aligned}$$

With suitable padding, both C_1 and C_2 have the same size. Thus, *Samp* satisfies the conditions needed for invoking the indistinguishability property of the obfuscator.

Now, we can describe the algorithm D, which takes as input τ as given above, and either the obfuscation of C_1, which is the program RSA Hash, or C_2, which is the program RSA Hash*. D then invokes the adversary, which makes requests for signatures on messages. D constructs the signatures $H(m)^d$, through its knowledge of d within τ. Finally, the attacker sends a forgery-message pair (σ^*, m^*) and wins if $\mathsf{Verify}(\mathrm{VK}, m^*, \sigma^*) = 1$. If the attacker wins, D outputs 1.

By construction, if D receives an obfuscation of C_1, then the probability that D outputs 1 is exactly the probability of the adversary winning in hybrid Hyb_0. On the other hand, if D receives an obfuscation of C_2, then the probability that D outputs 1 is the probability of the adversary winning in hybrid Hyb_1.

The lemma follows. ■

Lemma 8. If the RSA assumption holds, then the advantage of an poly-time algorithm in Hyb_3 is negligible.

Proof. We prove this lemma by giving a reduction to the RSA problem. To do so, we build algorithm $\mathcal{B}$.

$\mathcal{B}$ takes as input an RSA challenge (N, e, v) where N is the product of two (unknown) primes p, q, e is randomly chosen from $[1, \phi(N)]$ such that $\gcd(\phi(N), e) = 1$ and $v \in \mathbb{Z}_N^*$. Next, $\mathcal{B}$ calls $\mathrm{AdmSample}(1^\lambda, Q) \rightarrow u$, where Q is an upper bound on the number of queries made by the adversary. Then $\mathcal{B}$ chooses integers $(c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1})$ in the following way. For $i \in [n]$ and $b \in \{0,1\}$ if $b = u_i$ then first $y_{i,b}$ is chosen uniformly at random from all integers in $[1, N]$, and $c_{i,b} = e \cdot y_{i,b}$. Otherwise $y_{i,b}$ is chosen uniformly at random from all integers in $[1, N]$, and $c_{i,b} = e \cdot y_{i,b} + 1$. Finally, $\mathcal{B}$ creates the hash function $H(\cdot)$ as an obfuscation of the program RSA Hash* using the N and values $(c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1})$ above, and the value v in the RSA Hash* program will be the value v in the RSA challenge. All these steps together simulate the setup phase of Hyb_3. Now, it runs the attacker using the initial parameters generated above.

The attacker will then make at most Q signing queries each for a message m. We denote $m' = h(m)$ and the integer $\pi(m') = \prod_{i \in [n]} c_{i,m'_i}$. If $P_u(m) \neq 1$ $\mathcal{B}$ aborts and quits. Otherwise, $P_u(m) = 1$ and there exists an i where $e \mid c_{i,m'_i}$ and therefore $e \mid \pi(m')$. $\mathcal{B}$ can then compute the integer $t = \pi(m')/e$ and compute the (unique) signature on m as v^t.

Finally, the attacker will output an attempted forgery σ^* on some message m^* that is distinct from all the messages in the query phase. $\mathcal{B}$ first checks if the signature verifies and aborts if it does not. Next, it checks if $P_u(m^*) \neq 0$ and aborts if that is the case. Otherwise, $P_u(m^*) = 0$ and for all i we have $\gcd(e, c_{i,m'^*_i}) = 1$ and therefore $\gcd(e, \pi(m'^*)) = 1$. Following Shamir's theorem [Sha83], the attacker applies the Euclidean Algorithm to obtain integers α and β such that $\alpha \cdot e + \beta \cdot \pi(m'^*) = 1$. Therefore, since $(\sigma^*)^e = v^{\pi(m'^*)}$, it sets $z = v^\alpha \cdot (\sigma^*)^\beta$, and we have that $z^e = v^{\alpha e + \beta \pi(m'^*)} = v$. If σ^* was a successful forgery, then this value z is a solution to the RSA challenge.

We observe that by construction of $\mathcal{B}$, the probability of success of $\mathcal{B}$ is exactly the probability that the attacker succeeds in hybrid Hyb_3. Importantly, whenever $\mathcal{B}$ aborted, the attacker by the rules of Hyb_3 was not considered to be successful since his queries or forgery violated the partition. The lemma follows.

■

Pulling together these four lemmas immediately gives our the main theorem that the above RSA full domain hash signature scheme is (adaptively) secure.

5 Selectively Secure BLS Signatures

We now give a concrete construction for the hash function modeled as a random oracle in the Boneh-Lynn-Shacham (BLS) signature scheme. BLS signatures fall into a broad interpretation (see e.g., [Boy08]) of the full domain hash paradigm of Bellare and Rogaway.

Below we give the BLS signature scheme with a concrete hash function built from an indistinguishability obfuscator. We prove the signature scheme selectively secure based on the computational Diffie-Hellman problem in bilinear groups and a indistinguishability obfuscator.

On a technical level this selective proof of security follows a very similar structure to that of our selectively secure scheme from trapdoor functions from Section 3. The main difference is that here we deal with the mechanics of an algebraic bilinear group instead of a trapdoor function. We present the scheme for simplicity in terms of a symmetric bilinear group, however, we remark that moving to asymmetric groups is straightforward.

As in Section 3, we assume that there is a polynomial $\ell(\lambda)$ which denotes the length of messages to be signed; we denote this message space by $\mathcal{M} = \{0,1\}^{\ell(\lambda)}$. More generally, a collision-resistant hash function may be used to hash messages to this size.

- $\mathsf{Setup}(1^\lambda)$: The setup algorithm first runs the group generator on input 1^λ to produce a description of groups $\mathbb{G}, \mathbb{G}_T$ of prime order p along with generator $g \in \mathbb{G}$. These groups are related by a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Next, it chooses a random exponent $a \in \mathbb{Z}_p$. Then, the setup algorithm chooses a puncturable PRF key K for F where $F(K, \cdot) : \{0,1\}^{\ell(\lambda)} \to \mathbb{Z}_p$. Finally, it creates an obfuscation of the program BLS Selective Hash of Figure 5. The size of the program is padded to be the maximum of itself and the program BLS Selective Hash* of Figure 6. We refer to the obfuscated program as the function $H : \{0,1\}^\ell \to \mathbb{G}$, which acts as the random oracle type hash function in the BLS scheme.

  The verification key VK consists of the group descriptions $\mathbb{G}, \mathbb{G}_T$, the order p, the generator g and $A = g^a$ as well as the hash function $H(\cdot)$. The secret key is $a \in \mathbb{Z}_p$ as well as $H(\cdot)$.

- $\mathsf{Sign}(\mathrm{SK}, m \in \mathcal{M})$: The signature algorithm outputs $\sigma = H(M)^a \in \mathbb{G}$.

- $\mathsf{Verify}(\mathrm{VK}, m, \sigma)$ The verification algorithm tests if $e(\sigma, g) \overset{?}{=} e(A, H(m))$ and outputs accept if and only if this holds.

BLS Selective Hash

Constants: PRF key K, group generator $g \in \mathbb{G}$.
Input: Message m.

1. Output $g^{F(K,m)}$.

Figure 5: BLS Selective Hash

BLS Selective Hash*

Constants: Punctured PRF key $K(\{m^*\})$, $m^* \in \mathcal{M}$, $z^* \in \mathbb{G}$ and group generator $g \in \mathbb{G}$.
Input: Message m.

1. If $m = m^*$ output z^* and exit.
2. Output $g^{F(K,m)}$.

Figure 6: BLS Selective Hash*

Theorem 5. If our obfuscation scheme is indistingishuably secure, F is a secure punctured PRF, and the computational Diffie-Hellman problem holds in bilinear groups, then the above signature scheme is selectively secure.

We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original signature security game. We prove that a poly-time attacker's advantage must be negligibly close between each successive one. Then, we show that any poly-time attacker in the final experiment that succeeds in forging with non-negligible probability can be used to break the computational Diffie-Hellman assumption in bilinear groups.

- Hyb_0 : In the first hybrid the following game is played:

  1. The attacker selectively gives the challenger the message m^*.

2. The challenger runs the group generator to produce bilinear groups $\mathbb{G}, \mathbb{G}_T$ of order p with generator $g \in \mathbb{G}$. It then chooses a random exponent $a \in \mathbb{Z}_p$ for the secret key and sets $A = g^a$ as part of the verification key.
3. K is chosen as a key for the puncturable PRF.
4. The hash function $H(\cdot)$ is created as an obfuscation of the program BLS Selective Hash.
5. The attacker queries the sign oracle a polynomial number of times on messages $m \neq m^*$. It receives back $H(m)^a = A^{F(K,m)}$.
6. The attacker sends a forgery σ^* and wins if $\mathsf{Verify}(m^*, \sigma^*) = 1$.

- Hyb_1 : Is the same as Hyb_0 except we let $z^* = g^{F(K,m^*)}$ and let VK be the obfuscation of the program BLS Selective Hash* of Figure 6.
- Hyb_2 : Is the same as Hyb_1 except $z^* = g^t$ for t chosen uniformly at random in $\mathbb{Z}_p$.

Lemma 9. If our obfuscation scheme is indistinguishability secure, then the advantage of an poly-time algorithm in Hyb_0 is negligibly close to the advantage in Hyb_1.

Proof. We prove this lemma by giving a reduction to the indistinguishability security of the obfuscator. To do so, we must build the two algorithms $Samp$ and D.

$Samp(1^\lambda)$ behaves as follows: It invokes the adversary to obtain m^* and the adversary's state τ'. It runs the bilinear group setup on 1^λ to obtain the group descriptions $\mathbb{G}, \mathbb{G}_T$, order p and generator g. It then chooses a random $a \in \mathbb{Z}_p$ and sets $A = g^a$. It then chooses K as a key for the puncturable PRF. It sets $z^* = g^{F(K,m^*)}$. It sets $\tau = (m^*, z^*, g, p, A, K, \tau')$ and builds C_1 as the program for BLS Selective Hash, and C_2 as the program for BLS Selective Hash*.

Before describing D, we observe that by construction and the functionality preservation property of puncturable PRFs, the circuits C_1 and C_2 always behave identically on every input. Because of padding, both C_1 and C_2 have the same size. Thus, $Samp$ satisfies the conditions needed for invoking the indistinguishability property of the obfuscator.

Now, we can describe the algorithm D, which takes as input τ as given above, and either the obfuscation of C_1, which is the program BLS Selective Hash, or C_2, which is the program BLS Selective Hash*. D creates the verification key for the signature scheme with the obfuscated program as the hash function description. It then invokes the adversary on this verification key, and the adversary makes requests for signatures on messages $m \neq m^*$. For each such message, D constructs the signatures $H(m)^a = A^{F(K,m)}$, through its knowledge of K and A within τ. Finally, the attacker sends a forgery σ^* and wins if $\mathsf{Verify}(m^*, \sigma^*) = 1$. If the attacker wins, D outputs 1.

By construction, if D receives an obfuscation of C_1, then the probability that D outputs 1 is exactly the probability of the adversary winning in hybrid Hyb_0. On the other hand, if D receives an obfuscation of C_2, then the probability that D outputs 1 is the probability of the adversary winning in hybrid Hyb_1.

The lemma follows. ■

Lemma 10. If our confined PRF is secure, then the advantage of an poly-time algorithm in Hyb_1 is negligibly close to the advantage in Hyb_2.

Proof. We prove this lemma by giving a reduction to the pseudorandomness property at punctured points for punctured PRFs. To do so, we must build the algorithms A_1 and A_2.

$A_1(1^\lambda)$ simply invokes the adversary to obtain the challenge message m^* and state τ', and outputs the singleton set $S = \{m^*\}$ and $\tau = (1^\lambda, \tau')$.

A_2 obtains as input τ, the punctured key K_S, the singleton set $S = \{m^*\}$, and either a value $t^* = F(K, m^*)$ or a uniformly random value $t^* \in \mathbb{Z}_p$. Then, A_2 runs the group generator on 1^λ to obtain $(\mathbb{G}, \mathbb{G}_T, p, g)$, chooses a random $a \in \mathbb{Z}_p$, and sets $A = g^a$ to establish portions of VK and SK. Now given t^*, it can compute $z^* = g^{t^*}$. Note that this yields either the z^* value computed in hybrid Hyb_1 or in hybrid Hyb_2.

Since it knows K_S, now A_2 can obfuscate the program BLS Selective Hash*, and then execute the adversary and answer its signature queries using the punctured key K_S. Finally, A_2 outputs 1 if the adversary succeeds.

By construction, the pseudorandomness property for punctured PRFs implies the lemma. ■

Lemma 11. If the computational Diffie-Hellman assumption holds in bilinear groups, then the advantage of an poly-time algorithm in Hyb_2 is negligible.

Proof. We prove this lemma by giving a reduction to the hardness of the Computational Diffie-Hellman problem in the bilinear group $\mathbb{G}$. To do so, we build a CDH attacker $\mathcal{B}$.

$\mathcal{B}$ takes as input the tuple (g, g^a, g^b) for a group $\mathbb{G}$ of prime order p with a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. It sets $A = g^a$ as part of the verification key VK. The algorithm $\mathcal{B}$ then invokes the adversary to obtain m^*, and chooses a PRF key K and builds the punctured key $K(S)$ where $S = \{m^*\}$. It uses this key, together with VK and $z^* = g^b$, to obfuscate the program BLS Selective Hash*. It can then execute the adversary, and use its knowledge of $K(S)$ to answer all adversary signing queries. The adversary then terminates with an attempted forgery σ^* on message m^*. By the definition of the program BLS Selective Hash*, this forgery can only be valid if $\sigma^* = (z^*)^a = g^{ab}$. Thus if the adversary is successful, $\mathcal{B}$ can output σ^* as the solution to the CDH problem.

We observe that by construction of $\mathcal{B}$, the probability of success of $\mathcal{B}$ is exactly the probability that the attacker succeeds in hybrid Hyb_2. The lemma follows. ■

Pulling together these three lemmas immediately gives our the main theorem that the above full domain hash signature scheme is selectively secure.

6 Adaptively Secure BLS Signatures

We now give a hash function for BLS signatures that can be used to prove adaptive (or standard) security. Our proof structure will follow in a similar path to that of our adaptively secure RSA full domain hash signatures in Section 4. In particular, we will again apply an admissible hash function to partition the message space in our proof. At the same time, there are important distinctions and corresponding challenges that arise in this setting.

Again, for simplicity of exposition, we assume that there is a polynomial $\ell(\lambda)$ which denotes the length of messages to be signed; we denote this message space by $\mathcal{M} = \{0,1\}^{\ell(\lambda)}$ and we will often simply refer to the size of messages to be signed by ℓ. As in Section 4, we use a function $h : \{0,1\}^{\ell(\lambda)} \rightarrow \{0,1\}^{n(\lambda)}$, and an efficient randomized algorithm AdmSample that is *θ-admissible.*

Our construction is identical to that given in Section 6 with the exception of how the setup creates the hash function. The setup first chooses uniformly at random $(c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1})$ each in $\mathbb{Z}_p$. Then it obfuscates the program BLS Adaptive Hash of Figure 7 where the size of the program is padded to be the maximum of itself and the program BLS Adaptive Hash* of Figure 8. The obfuscated program is used as the function $H : \{0,1\}^{\ell(\lambda)} \rightarrow \mathbb{G}$, which acts as the random oracle type hash function in the BLS scheme.

Our proof of security relies on indistinguishability obfuscation and our Diffie-Hellman Inversion equivalent assumption. Namely, that given $g, g^a, g^{a^2}, \ldots, g^{a^n} \in \mathbb{G}$, it is hard to compute $g^{a^{n+1}}$.

BLS Adaptive Hash

Constants: Bilinear group $\mathbb{G}$ with generator g and exponents $(c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1})$ each in $\mathbb{Z}_p$.
Input: Message $m \in \{0,1\}^{\ell}$.

1. Compute $m' = h(m)$.
2. Output $g^{\prod_{i=1,\ldots,n} c_{i,m'_i}}$.

Figure 7: BLS Adaptive Hash

BLS Adaptive Hash*

Constants: Bilinear group $\mathbb{G}$, elements $g, g^a, g^{a^2}, \ldots, g^{a^n} \in \mathbb{G}$, for $i \in [n], b \in \{0,1\}$ exponents $y_{i,b} \in \mathbb{Z}_p$ and $u \in \{0,1\}^n$.
Input: Message $m \in \{0,1\}^\ell$.

1. Compute $m' = h(m)$.
2. Let $\mu(m)$ be the set i such that $m'_i \neq u_i$. The algorithm computes the set size $|\mu(m)|$.
3. Output $(g^{a^{|\mu(m)|}})^{\prod_{i=1,\ldots,n} y_{i,m'_i}}$.

Figure 8: BLS Adaptive Hash*

Theorem 6. If our obfuscation scheme is indistinguishability secure and the Diffie-Hellman Inversion assumption holds in bilinear group $\mathbb{G}$, the above signature scheme is existentially unforgeable against chosen message attacks.

We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original signature security game. As in Section 4, in the first hybrid step we do a "partitioning" of the message space. After the first proof step, we prove that any poly-time attacker's advantage must be negligibly close between each successive hybrid experiment. We finally show that any poly-time attacker in the final experiment that succeeds with non-negligible probability can be used to break our assumption that is equivalent to the Diffie-Hellman Inversion assumption. See Section 2.5 for more on these assumptions.

- Hyb_0 : In the first hybrid, the following game is played:
  1. The challenger runs the group generator to produce bilinear groups $\mathbb{G}, \mathbb{G}_T$ of order p with generator $g \in \mathbb{G}$. It then chooses a random exponent $a \in \mathbb{Z}_p$ for the secret key and sets $A = g^a$ as part of the verification key.
  2. It chooses uniformly at random $(c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1})$ each in $\mathbb{Z}_p$.
  3. The hash function $H(\cdot)$ is created as an obfuscation of the program BLS Adaptive Hash.
  4. The attacker queries the signing oracle at most Q times on messages $m_1, \ldots, m_Q$. In its ith query, it receives back $H(m_i)^a$.
  5. The attacker finally chooses a message m^*, sends a forgery σ^*, and wins if $\mathsf{Verify}(\mathrm{VK}, m^*, \sigma^*) = 1$.
- Hyb_1 : Is the same as Hyb_0 except the challenger begins by sampling a string $u \in (\{0,1,\perp\})^n$ by calling $\mathrm{AdmSample}(1^\lambda, Q) \to u$, where Q is an upper bound on the number of queries made by the adversary (this could, for example, be the running time of the adversary). At the end of the experiment, the attacker is only considered to be successful if both $\mathsf{Verify}(\mathrm{VK}, m^*, \sigma^*) = 1$ and $P_u(m^*) = 0$ and for all messages m_i queried $P_u(m_i) = 1$. If the attacker is successful but this condition is not satisfied, we say that the hybrid "aborts."
- Hyb_2 : Is the same as Hyb_1 except the for the following modification. The challenger first chooses exponents $(c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1})$ in the following way. For $i \in [n], b \in \{0,1\}$ it chooses random $y_{i,b} \in \mathbb{Z}_p$ and then sets
$$c_{i,b} = \begin{cases} y_{i,b} & \text{if } b = u_i \\ a \cdot y_{i,b} & \text{if } b \neq u_i \end{cases}$$
- Hyb_3 : Is the same as Hyb_2 except the challenger creates the hash function $H(\cdot)$ as an obfuscation of the program BLS Adaptive Hash*.

Lemma 12. Consider a attacker that makes at most a polynomial of queries $Q = Q(\lambda)$ in Hyb_0. If the advantage of an attacker in Hyb_0 is $\epsilon(\lambda)$, then the advantage of the attacker in Hyb_1 will be at least $\epsilon(\lambda)/\theta(Q)$. In particular, any poly-time attacker with non negligible advantage in Hyb_0 will also have non-negligible advantage in Hyb_1.

Proof. The lemma follows immediately from the function h satisfying the definition of a θ-admissibility, since the only independent choice of $u \leftarrow \text{AdmSample}(1^\lambda, Q)$ determines whether or not the hybrid aborts. (This argument is identical to the corresponding one in the proof of Lemma 5 of Section 4.) ■

Lemma 13. The advantage of any poly-time algorithm in Hyb_1 is the same as its advantage in Hyb_2.

Proof. The two hybrid experiment are equivalent as all $c_{i,b} \in \mathbb{Z}_p$ values are still chosen uniformly at random in both hybrids. (The step from Hyb_1 to Hyb_2 is a notational reorganization to set up the next proof step.) ■

Lemma 14. If our obfuscation scheme is indistinguishability secure, then the advantage of any poly-time algorithm in Hyb_2 is negligibly close to the advantage in Hyb_3.

Proof. We prove this lemma by giving a reduction to the indistinguishability security of the obfuscator. To do so, we must build the two algorithms *Samp* and D.

Samp(1^λ) behaves as follows: It invokes the adversary to obtain the adversary's state τ'. It runs the bilinear group setup to obtain $\mathbb{G}, \mathbb{G}_T, p, g$ and then chooses a random $a \in \mathbb{Z}_p$. For $i \in [n], b \in \{0,1\}$ it chooses random $y_{i,b} \in \mathbb{Z}_p$ and then sets

$$c_{i,b} = \begin{cases} y_{i,b} & \text{if } b = u_i \\ a \cdot y_{i,b} & \text{if } b \neq u_i \end{cases}$$

It samples a string $u \in (\{0,1,\perp\})^n$ by calling $\text{AdmSample}(1^\lambda, Q) \rightarrow u$, where Q is an upper bound on the number of queries made by the adversary. It sets $\tau = (\mathbb{G}, g, a, (c_{1,0}, c_{1,1}), \ldots, (c_{n,0}, c_{n,1}), (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1}), u, \tau')$ and builds C_1 as the program for BLS Adaptive Hash, and C_2 as the program for BLS Adaptive Hash*.

Before describing D, we observe that by construction, the circuits C_1 and C_2 always behave identically on every input. To show program equivalence, note that for all m', we have that

$$g^{\prod_i c_{i,m'_i}} = g^{a^{|\mu(m')|} \cdot \prod_i y_{i,m'_i}} = (g^{a^{|\mu(m')|}})^{\prod_i y_{i,m'_i}}.$$

With suitable padding, both C_1 and C_2 have the same size. Thus, *Samp* satisfies the conditions needed for invoking the indistinguishability property of the obfuscator.

Now, we can describe the algorithm D, which takes as input τ as given above, and either the obfuscation of C_1, which is the program BLS Adaptive Hash, or C_2, which is the program BLS Adaptive Hash*. D then invokes the adversary, which makes requests for signatures on messages. D constructs the signatures $H(m)^a$, through its knowledge of a within τ. Finally, the attacker sends a forgery-message pair (σ^*, m^*) and wins if $\mathsf{Verify}(\text{VK}, m^*, \sigma^*) = 1$. If the attacker wins, D outputs 1.

By construction, if D receives an obfuscation of C_1, then the probability that D outputs 1 is exactly the probability of the adversary winning in hybrid Hyb_2. On the other hand, if D receives an obfuscation of C_2, then the probability that D outputs 1 is the probability of the adversary winning in hybrid Hyb_3.

The lemma follows. ■

Lemma 15. If the Diffie-Hellman Inversion Assumption holds in bilinear group $\mathbb{G}$, then the advantage of any poly-time algorithm in Hyb_3 is negligible.

Proof. We prove this lemma by giving a reduction to the Diffie-Hellman Inversion problem. To do so, we build algorithm $\mathcal{B}$.

$\mathcal{B}$ takes as input a n-DHI challenge $(g, g^a, g^{a^2}, \ldots, g^{a^n})$ from a bilinear group $\mathbb{G}$ of prime order p, where n is the same as the output length of the admissible hash function $h(\cdot)$. Next, $\mathcal{B}$ calls $\text{AdmSample}(1^\lambda, Q) \rightarrow u$, where Q is an upper bound on the number of queries made by the adversary. For $i \in [n], b \in \{0,1\}$, it chooses random $y_{i,b} \in \mathbb{Z}_p$.

Finally, $\mathcal{B}$ creates the hash function $H(\cdot)$ as an obfuscation of the program BLS Adaptive Hash* using the DHI challenge values, the values $(y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$ above and u. All these steps together simulate the setup phase of Hyb_3. Now, it runs the attacker using the initial parameters above, including $A = g^a$.

The attacker will then make at most Q signing queries each for a message m. We denote $m' = h(m)$. If $P_u(m) \neq 1$, $\mathcal{B}$ aborts and quits. Otherwise, $P_u(m) = 1$ and there exists an i where $m'_i = u_i$, meaning that the hash of m' will contain a power of a that is strictly less than n. Thus, the signature can be formed using only knowledge of the DHI input and the $y_{i,b}$ values.

Finally, the attacker will output an attempted forgery σ^* on some message m^* that is distinct from all the messages in the query phase. $\mathcal{B}$ first checks if the signature verifies and aborts if it does not. Next, it checks if $P_u(m^*) \neq 0$ and aborts if that is the case. Otherwise, $P_u(m^*) = 0$ and for all i we have $h(m^*)_i \neq u_i$. This means that the hash of m^* will be g^{a^n} raised to some known product of $y_{i,b}$ values. The signature therefore contains $g^{a^{n+1}}$ raised to some known product of $y_{i,b}$ values, since signatures contain one more factor of a in the exponent than their corresponding hash values. This value can be recovered by taking the proper root of the signature, i.e., $(\sigma^*)^{1/\prod_i y_{i,h(m^*)_i}} = g^{a^{n+1}}$, and thus if σ^* was a successful forgery, then this root of the signature is a solution to the DHI challenge.

We observe that by construction of $\mathcal{B}$, the probability of success of $\mathcal{B}$ is exactly the probability that the attacker succeeds in hybrid Hyb_3. Importantly, whenever $\mathcal{B}$ aborted, the attacker by the rules of Hyb_3 was not considered to be successful since his queries or forgery violated the partition. The lemma follows. ∎

Pulling together these four lemmas immediately gives our the main theorem that the above BLS signature scheme is (adaptively) secure.

7 Extensions to Boneh-Franklin IBE and Aggregate Signatures

Boneh-Franklin IBE We can adapt our techniques for proving security of BLS signatures to the Boneh-Franklin [BF01] Identity-Based Encryption system. BLS signatures directly correspond to IBE private keys in the BF scheme. The proof for the BF adapts with a few minor changes:

- For proving BF selectively secure we can use the decision Bilinear Diffie-Hellman assumption.
- The second random oracle in the BF scheme can be replaced with an extractor.
- For proving adaptive security we use the following assumption. Namely that given $g, g^s, g^a, g^{a^2}, \ldots, g^{a^n}$ it is hard to distinguish $e(g,g)^{a^{n+1}s}$ from a random group element in $\mathbb{G}_T$. We note this assumption is weaker than the decision Bilinear Diffie-Hellman Exponent assumption [BGW05].

BLGS Aggregate Signatures Boneh, Gentry, Lynn and Shacham [BGLS03] showed that the BLS signatures are aggregateable. Our results on BLS signatures translate immediately.

Acknowledgments

We thank Mihir Bellare for discussions relating to the origins and terminology of full domain hash signatures. We thank Dan Boneh for pointing out the equivalence of our assumption used in Section 6 to the Diffie-Hellman Inversion assumption. We also thank Dennis Hofheinz for clarifications on admissible hash functions and pointing us to the simplified version we used.

References

[BB04a] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459, 2004.

[BB04b] Dan Boneh and Xavier Boyen. Short signatures without random oracles. *IACR Cryptology ePrint Archive*, 2004:171, 2004.

[BBP04] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In *EUROCRYPT*, pages 171–188, 2004.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, 2001.

[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 *CRYPTO*, pages 1–18, 2001.

[BGI$^+$12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[BGI13] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. *IACR Cryptology ePrint Archive*, 2013:401, 2013.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, pages 416–432, 2003.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO*, pages 258–275, 2005.

[BHK13] Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Instantiating random oracles via uces. *IACR Cryptology ePrint Archive*, 2013:424, 2013.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT*, pages 514–532, 2001.

[Boy08] Xavier Boyen. A tapestry of identity-based encryption: practical frameworks compared. *IJACT*, 1(1):3–21, 2008.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with rsa and rabin. In *EUROCRYPT*, pages 399–416, 1996.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. *IACR Cryptology ePrint Archive*, 2013:352, 2013.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *STOC*, pages 209–218, 1998.

[CHK07] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. *J. Cryptology*, 20(3):265–294, 2007.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, pages 13–25, 1998.

[DOP05] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In *CRYPTO*, pages 449–466, 2005.

[FHPS13] Eduarda S. V. Freire, Dennis Hofheinz, Kenneth G. Paterson, and Christoph Striecks. Programmable hash functions in the multilinear setting. *IACR Cryptology ePrint Archive*, 2013:354, 2013.

[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 *FOCS*, 2013.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the fiat-shamir paradigm. In *FOCS*, pages 102–113, 2003.

[HK12] Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. *J. Cryptology*, 25(3):484–527, 2012.

[HSW13] Susan Hohenberger, Amit Sahai, and Brent Waters. Full domain hash from (leveled) multilinear maps and identity-based aggregate signatures. In *CRYPTO*, 2013.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. *IACR Cryptology ePrint Archive*, 2013:379, 2013.

[KS98] B. Kaliski and J. Staddon. Pkcs #1: Rsa cryptography specifications version 2.0, 1998.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[Sha83] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Trans. Comput. Syst.*, 1(1):38–44, 1983.

[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/`.

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.