Realizing Hash-and-Sign Signatures under Standard Assumptions

Susan Hohenberger*
Johns Hopkins University

Brent Waters†
University of Texas at Austin

January 12, 2009

Abstract

Currently, there are relatively few instances of "hash-and-sign" signatures in the standard model. Moreover, most current instances rely on strong and less studied assumptions such as the Strong RSA and q-Strong Diffie-Hellman assumptions.

In this paper, we present a new approach for realizing hash-and-sign signatures in the standard model. In our approach, a signer associates each signature with an index i that represents how many signatures that signer has issued up to that point. Then, to make use of this association, we create simple and efficient techniques that restrict an adversary which makes q signature requests to forge on an index no greater than $2^{\lceil \lg(q) \rceil} < 2q$. Finally, we develop methods for dealing with this restricted adversary.

Our approach requires that a signer maintains a small amount of state — a counter of the number of signatures issued. We achieve two new realizations for hash-and-sign signatures respectively based on the RSA assumption and the Computational Diffie-Hellman assumption in bilinear groups.

1 Introduction

Digital signatures are a fundamental cryptographic primitive and a key building block in many larger systems. Typically, known constructions fall into one of two categories: the "tree"-based approach or the "hash-and-sign" approach. This latter paradigm generally yields more efficient constructions and shorter signatures, and represents what practitioners have come to expect. While many realizations of hash-and-sign signatures exist in the random oracle model (e.g., [16, 32, 27, 4, 28, 7, 19, 18]), efficient schemes in the standard model are rare. Moreover, random oracle model schemes can be based on well-studied assumptions such as the discrete logarithm problem, Computational Diffie-Hellman and RSA. However, known standard model schemes are often based on much stronger assumptions, such as Strong RSA [17, 13], q-Strong Diffie-Hellman [6] and LRSW [10].[1] These assumptions allow the adversary a significant amount of flexibility in his ability to win, such

*Supported by National Science Foundation grant CNS-0716142 and a Microsoft New Faculty Fellowship.

†Supported by NSF CNS-0524252, CNS-0716199, CNS-0749931; the US Army Research Office under the CyberTA Grant No. W911NF-06-1-0316; and the U.S. Department of Homeland Security under Grant Award Number 2006-CS-001-000001. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security. Portions of this work were done while this author was at SRI International.

[1] One recent exception is the signature scheme due to Waters [33], which is provably secure under the CDH assumption in bilinear groups. However, this scheme suffers from a large public key size.

as allowing him to choose a signing exponent (Strong RSA), compute a value relative to a chosen constant (q-Strong Diffie Hellman), or choose a random base value (LRSW). In each case, there are *many* possible correct answers to the adversary's challenge; in fact, exponentially many. This stands in high contrast to weaker, more studied assumptions, such as the discrete logarithm problem, Computational Diffie-Hellman and RSA, where there is only *one* correct answer for a given challenge. These assumptions which restrict the adversary to a single correct response seem inherently more reliable than their flexible counterparts. Thus, an important direction, in our opinion, is to push forward to practical, standard model schemes under standard assumptions.

One challenging aspect is that the security definition of signatures [21] inherently allows the adversary a great deal of flexibility; she wins if she outputs a forgery on *any* message not previously signed. Likewise, most existing "hash-and-sign" standard model schemes inherently enable the adversary a good deal of flexibility on which forgeries it can output and then the security is based on the hardness of a problem where there are *many* possible solutions. For example, consider the construction of Cramer and Shoup [17, 13]; since the legitimate signer chooses a random prime from an exponentially large range, any proof must consider a forger that has the same flexibility in choosing the exponent and therefore the reduction is to the Strong RSA assumption (i.e, given (N, y), it is hard to produce *any* pair (e, x) such that $e > 1$ and $x^e \equiv y \mod N$).

In this work, we present a new avenue: design the scheme in such a way that *enforces* that any adversary output forgeries in some small set of categories that roughly grows with the number of signatures created so far. (E.g., A category could correspond to an RSA exponent used for verification in a prior signature.) Once the forger is restricted to a small set of categories, the simulator can guess where to program the challenge within this set. Alternatively, one can view our approach as restricting the adversary to a small forgery set and then employing *selectively-secure* techniques. The primary contribution of this work is the new method for restricting a forger.

Our Approach. Let us give the intuition behind our two constructions. At the core of our method, we associate with each signature an index i representing the number of signatures previously issued by the signer. The actual signer will only issue one signature per index. Roughly, the idea is to *efficiently* force the adversary to forge on a previously seen index value. To restrict the adversary, each signature is comprised of two logical components: a "core" signature on the message under index i and a second component that bounds the highest index number that the adversary might use. The most naive method would be to create a separate signature that signs the current index being used; however, this would itself demand a secure signature scheme and lead to a circularity. To avoid this, the second component for a signature on index i will be a "signature" on $\lceil \lg(i) \rceil$. If we allow at most 2^λ signatures, then there are at most λ possible values for this second component and realizing it becomes simple. Moreover; the set of "allowable" indices is at most a factor of 2 times the number of signatures given out so far. It follows that any adversary must forge on a index set of roughly the same size as the number of signatures he has seen (or break the second component). Once we apply these techniques to force the adversary into this small index set, we are in a position to create a system based on weaker assumptions.

Let us illustrate this by describing a simplified version of our RSA-based construction. Let N be a Blum-Williams integer. The signer publishes a modulus N, a random value $v \in \mathbb{Z}_N^*$ and a hash function that enumerates a sequence of primes, i.e., let $H(i) = e_i$. To generate a signature using index i on message m, the signer creates a "core" signature on m using the signing exponent e_i^{-1} and then also gives out i and the $\lceil \lg(i) \rceil$-th square root of v. This ensures that an adversary

that makes at most q queries *must* sign using one of the first $2^{\lceil \lg(q) \rceil} < 2q$ values of e_i (i.e., the output of $H(j)$ on $j = 1$ to $2^{\lceil \lg(q) \rceil}$); otherwise, the adversary can take square roots and thereby factor N. Now that the adversary is restricted to forging using a small set of e_i values, we can use a combination of previous techniques and new ideas to reduce from the standard RSA assumption.

Outline. We realize two new hash-and-sign signatures under the RSA assumption and the Computational Diffie-Hellman assumption in bilinear groups, in Sections 3 and 4 respectively. In Section 5, we discuss how to manage the signer's state in practice, including across multiple machines.

1.1 Related Work

The related work on designing secure signature schemes is vast and long standing. We provide only a brief summary.

Tree-Based. Many of the earliest provably-secure constructions used the design paradigm of a tree. Here a bound on the number of signatures to be issued is first established, and then the efficiency of the signatures (i.e., their size and the size of the public key) is in some way proportional to this bound. From general assumptions, a series of works including Bellare-Micali [2], Naor-Yung [26], and Rompel [31] established that signatures can be based on one-way functions. From general and concrete assumptions, another series of works sought more efficient solutions, such as those of Goldwasser-Micali-Rivest [21], Goldreich [20], Merkle [23], Dwork-Naor [15], Cramer-Damgård [11, 12] and many more. While these works are fundamental to our understanding of provably secure signatures, the tree-based constructions are often passed over in practice due to the computation and memory requirements.

Hash-and-Sign. In the search for more efficient constructions, many schemes in the random oracle model were proposed, such as those of El Gamal [16], Schnorr [32], Okamoto [27], Bellare-Rogaway [4], Pointcheval-Stern [28] and more recent short signatures by Boneh-Lynn-Shacham [7], signatures with tight-reductions to Diffie-Hellman by Goh-Jarecki-Katz-Wang [19], and the recent lattice-based signatures of Gentry-Peikert-Vaikuntanathan [18]. Unfortunately, the schemes are only known to be secure relative to the random oracle heuristic.

Drawing closer to our objective, some prior works have explored secure hash-and-sign signatures in the standard model. In 1999, Gennaro, Halevi and Rabin [17] introduced the first hash-and-sign construction secure in the standard model; its security depends on the Strong RSA assumption. Subsequent works also based on Strong RSA of Cramer-Shoup [13] and Camenisch-Lysyanskaya [9] improved the efficiency and added efficient protocols, respectively. (We will make use of a key reduction technique due to Cramer and Shoup later on.)

More recent works pushed for shorter signatures in the standard model, moving away from Strong RSA to more complex bilinear assumptions. Two such examples are the Boneh-Boyen [6] signatures based on q-Strong Diffie-Hellman (i.e., given a generator g of prime order p and the tuple $(g^x, g^{x^2}, \ldots, g^{x^q})$, it is hard to compute $(c, g^{1/(x+c)})$ for any $c \in \mathbb{Z}_p^*$) and the Camenisch-Lysyanskaya [10] signatures based on the interactive LRSW assumption.

While these standard model schemes are useful and efficient, their security depends on strong assumptions. In Strong RSA, q-Strong Diffie-Hellman and LRSW, there are *many* correct answers to any given challenge, allowing the adversary a significant amount of flexibility. This stands in sharp contrast to mild and restricted assumptions such as RSA and CDH.

To our knowledge, the only example prior to this work of a practical signature scheme secure in the standard model and under a mild complexity assumption is due to Waters [33], whose scheme is based on CDH in bilinear groups. The drawback of Waters' scheme compared to our scheme under the same assumption in Section 4 is that the public key requires $O(\lambda)$ group elements, where λ is the security parameter, whereas our public key requires $O(1)$ group elements. There exist variants of the Waters scheme (e.g., [25]) offering tradeoffs between the public key size and the concrete security level, but the asymptotic behavior remains the same.

Interpretting our Results. In this work, we limit ourselves to the standard practice of polynomial-time reductions. We note that if we allowed super-polynomial reductions it seems possible to interpret the Gennaro-Halevi-Rabin [17] and Cramer-Shoup [13] solutions as provably secure under ordinary RSA and the selectively-secure signatures of Boneh-Boyen [5] (as derived from their selectively-secure identity-based encryption scheme) as provably secure under CDH. Indeed, one alternative way of viewing our techniques is as a method for restricting a signature adversary so that selectively-secure schemes become (fully) adaptively-secure.

One can also view our results as a step toward realizing practical, standard model signatures under standard assumptions in a *stateless* manner. We remind the reader that many of the early tree-based signatures, such as the GMR signatures [21], also required the signer to keep a counter on the number of signatures issued. Subsequently, Goldreich [20] showed how to remove this dependence on state. We believe that a parallel outcome is possible here.

2 Background

2.1 Signature Schemes

Since we consider stateful signers, we slightly alter the signature algorithm specifications as follows:

KeyGen(1^λ) : the key generation algorithm outputs a keypair $(\mathrm{PK}, \mathrm{SK})$ and an initial state s.

Sign(SK, s, M) : the signing algorithm takes in a secret key SK, a state s, and a message M, and produces a signature σ.

Verify(PK, M, σ)**:** the verification algorithm takes in a public key PK, a message M, and a purported signature σ, and returns 1 if the signature is valid and 0 otherwise.

We use the standard security notion of *existential unforgeability with respect to chosen-message attacks* as formalized by Goldwasser, Micali and Rivest [21]. Here the adversary is given the public key and access to a signing oracle. The adversary is considered to be successful if she is able to produce a valid signature on any message not queried to the oracle.

2.2 Chameleon Hash Functions

A chameleon hash function $H(m, r)$ has the usual collision-resistant hash properties with the additional feature that, given some special trapdoor information, any target y and any message m', it is possible to efficiently find a value r' such that $H(m', r') = y$. Chameleon hash functions were first formalized by Krawczyk and Rabin [22], who also presented a discrete-logarithm-based construction, derived from the chameleon commitments of Boyar et al. [8]. We employ this hash in Section 4 for our CDH-based signatures. In our RSA-based signatures in Section 3, we can employ any chameleon hash function. We note that secure constructions exist in the standard model under

the discrete-logarithm assumption [22], the hardness of factoring [22], and the RSA assumption [1]. See Appendix A for more on RSA-based chameleon hashes.

2.3 RSA Assumption and other Facts

We begin by recalling (one of the) standard versions of the RSA assumption [30].

Assumption 2.1 (RSA). *Let k be the security parameter. Let positive integer N be the product of two k-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 remind the reader that in the **Strong RSA** assumption the adversary is given (N, y) and succeeds by producing any integer pair (e, x) such that $e > 1$ and $x^e \equiv y \mod N$. The standard RSA version is much more restrictive on the adversary.

We will make use of the following additional facts.

In our RSA-based scheme, we will require a primality test. Fortunately, for our purposes, it will be sufficient to use the efficient Miller-Rabin test [24, 29].

Lemma 2.2 (Cramer-Shoup [13]). *Given $x, y \in \mathbb{Z}_n$ together with $a, b \in \mathbb{Z}$ such that $x^a = y^b$ and $\gcd(a, b) = 1$, there is an efficient algorithm for computing $z \in \mathbb{Z}_n$ such that $z^a = y$.*

Later on, we will choose our RSA moduli from the set of Blum-Williams integers, since we will require that each square has a unique square root which is itself a square. Formally, we will use:

Lemma 2.3 (Bellare-Miner [3]). *Suppose n is a Blum-Williams integer. Suppose $a, a_1, \ldots, a_t \in \mathbb{Z}_n^*$ and a is a square modulo n. Suppose $x, x_1, \ldots, x_t$ are integers such that $x_1, \ldots, x_t > x \geq 0$. Suppose*

$$a^{2^x} = \prod_{j=1}^{t} a_j^{2^{x_j}} \mod n \text{ , then } a = \prod_{j=1}^{t} a_j^{2^{x_j - x}} \mod n.$$

Theorem 2.4 (Prime Number Theorem). *Define $\pi(x)$ as the number of primes $\leq x$. For $x > 1$,*

$$\pi(x) > \frac{x}{\ln x}.$$

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 $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ 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 2.5 (Computational Diffie-Hellman [14]). *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:*

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

3 Our RSA Realization

In our later CDH construction, the signer's state of i will directly correspond to the ith signature. This will not be true here. In our simplified scheme in the introduction, we assumed the existence of a function that maps states to different prime exponents. Our first challenge is to realize this function in a way that allows us a means in the proof for embedding our RSA challenge exponent. Our realization of this function, denoted H below, will require that we skip over some states. Tantamount to our success will be the ability to maintain a correct distribution in our reduction.

To gain some intuition into the construction, let us begin by describing a *publicly-computable* hash function $H : \mathbb{Z} \to \{0,1\}^k$, which will be used to link exponents to states. Let $F : \mathbb{Z} \to \{0,1\}^k$ be a pseudorandom function family. Next, let c be a random value in $\{0,1\}^*$. For a random PRF key K, we define our corresponding hash function as $H_K(x) := c \oplus F_K(x)$.

While it is unusual to publicly release a PRF key, we do so because we only require some weaker properties from our hash function for which this construction will be sufficient. Specifically, we require that the hash function H_K: (1) outputs large primes with sufficient probability, and (2) on the first polynomial inputs to the hash, all prime outputs are distinct with high probability. We will later show that if the function H_K does not meet these requirements then F could not have been a PRF family.

Let us now turn to how the hash function H_K is used in the system. The signer keeps state as before and when signing with state s, if $H_K(s)$ is not prime, the signer will skip over it and increment its state until it reaches some s' such that $H_K(s')$ is prime. It will then sign using this prime and state s'. Thus, it will be important to guarantee that the signer not have to skip over too many indices when issuing signatures.

Now, we present our core construction and then remark on some different possible optimizations. The construction here already reduces the public key and signature size by one element in Z_N^* over the simplified RSA construction described in the introduction.

3.1 RSA Construction

Setup(1^λ) The setup algorithm chooses a Blum-Williams integer N, such that $2^\ell < \phi(N) < 2^{\ell+2}$, where ℓ is another security parameter derived from 1^λ. It then chooses two random quadratic residues $u, h \in \mathrm{QR}_N$.

Next, it establishes a hash function $H : \mathbb{Z} \to \{0,1\}^\ell$ by choosing a random key K for the PRF function $F : \mathbb{Z} \to \{0,1\}^\ell$, a random $c \in \{0,1\}^\ell$, and defining $H_K(x) = c \oplus F_K(x)$.

It then publishes the parameters L of some Chameleon Hash scheme $\mathrm{ChamHash} : \{0,1\}^{\ell'} \times \{0,1\}^{\ell''} \to \{0,1\}^{\frac{2\ell}{3}}$. (We note that such schemes in the standard model exist under the hardness of factoring [22] and RSA [1]; see Appendix A.)

The public key consists of

$$N, u, h, c, K, L.$$

Note, anyone can compute $H_K()$ using these parameters. The setup algorithm sets its state counter $s = 0$ and keeps the factorization of N as the secret key SK.

Sign(SK, $s, M \in \{0,1\}^{\ell'}$) The signer first increments its counter s by one as $s = s + 1$. The algorithm then chooses a random $r \in \{0,1\}^{\ell''}$ from the appropriate range dictated by the choice of

ChamHash. It then computes $x = \text{ChamHash}(M, r)$. Next, it checks if $H_K(s)$ is a prime. If not it increments $s = s + 1$ and tries again until $e_s = H_K(s)$ is a prime. Then the signer computes:

$$B = (u^x h)^{(\frac{1}{2})^{\lceil \lg(s) \rceil}} \mod N$$

Note that we abuse notation here when taking square roots. When working modulo Blum-Williams integers, let $X^{\frac{1}{2}}$ represent the unique square root of X which is itself also a square. (See Lemma 2.3.) The signature is output as:

$$\sigma_1 = B^{\frac{1}{e_s}}, \quad r, \quad s.$$

Conceptually, s is an index, but we will skip over many s values where $H_K(s)$ is not a prime.

Verify$(\text{PK}, M, \sigma = (\sigma_1, r, i))$ The verification algorithm first makes sure that $i < 2^\lambda$. If it is greater, then it rejects. Second, the verifier checks that $H_K(i)$ is a prime. If not, it rejects.

Next, it squares σ_1 a total of $\lceil \lg(s) \rceil$ times yielding the value $Y = (\sigma_1)^{2^{\lceil \lg(s) \rceil}}$. Finally, it computes $x = \text{ChamHash}(M, r)$ and $e_i = H_K(i)$, and rejects unless it verifies that

$$Y^{e_i} \equiv (u^x h) \mod N.$$

3.1.1 Comments

The above scheme is geared to showcase the main ideas, however, one might consider different variants that allow for faster signature generation and faster verification. One area for improvement is in the way that prime exponents are generated and linked to states.

As a first variant, instead of having the signer skip over an index i if $H_K(i)$ is not prime (and thus, update and write to memory a state change), consider a scheme that allows the signer to search for a prime in a small range around the value $H_K(i)$. This option would require a more detailed analysis of the probability of a collision among the prime exponents used as well as a more complicated method for plugging in the RSA challenge.

As a second variant, consider a scheme that uses the first q primes starting with 3 to sign q messages. This variant would enable both faster generation (via a prime number seive to find the primes) and faster verification since we'd be using small prime exponents. Unfortunately, this appears to require a reduction from an assumption different than standard RSA; in particular, one might consider reducing from an assumption that the adversary can't take a root chosen randomly from the first q odd prime roots. For any polynomial q, this assumption is weaker than Strong RSA; however, we cannot reduce it to the standard RSA assumption.

A third avenue for optimization, focusing on the size of N and the chameleon hash parameters, is to note that our setting of $\{0,1\}^{\frac{2\ell}{3}}$ as the chameleon hash range is somewhat arbitrary. It can be set to any constant fraction of ℓ bits or any range R such that for a random prime $e \in \{0,1\}^\ell$ the probability that $e \notin R$ is non-negligible (we achieve $e \notin \{0,1\}^{\frac{2\ell}{3}}$ with high probability). In other words, there can be a tradeoff here between the size of the parameters and the concrete security. We also remind the reader that one can enlarge the domain of a chameleon hash by first applying a normal collision-resistant hash function [22].

A fourth avenue for optimization, this time focusing on the number of elements in the public key, is to find a method for directly embedding the chameleon hash function into the signature itself (as we do in our CDH scheme in Section 4).

3.2 Proof of Security

Theorem 3.1. *If the RSA assumption holds when N is a Blum-Williams integer, then the above construction is a secure signature scheme.*

Proof. Our reduction will only work on certain types of RSA challenges. We first describe this challenge set and then describe the reduction.

Our reduction will "throw out" all RSA challenges (N, e^*, y) where e^* is *not* an odd prime less than 2^ℓ. Fortunately, good challenges will occur with polynomial probability. By construction, $\phi(N) < 2^{\ell+2}$. We also know, by Theorem 2.4, that the number of primes $\leq 2^\ell$ is $\geq \frac{2^\ell}{\ell}$. Thus, a loose bound on the probability of e^* being a prime in the proper range is $(\frac{2^\ell}{\ell})/2^{\ell+2} = \frac{1}{4\ell}$.

Now, we describe the reduction. Suppose we have an adversary that makes at most $q(\lambda)$ queries where $q()$ is a polynomial. (We say q queries where it is clear from context.) We show that this adversary breaks RSA, on challenges (N, e^*, y) where N is a Blum-Williams integer and e^* is an odd prime $< 2^\ell$. An adversary can have two types of forgeries. Let x be the highest index on which the adversary obtains a signature from the signer (i.e., the index at which the qth prime appears).

Type I The adversary forges for a message with index i greater than $2^{\lceil \lg(x) \rceil}$.

Type II The adversary forges for a message with index i less than or equal to $2^{\lceil \lg(x) \rceil}$.

In Lemma 3.2, we show that a type I adversary can be used to break factoring with a loss of a λ factor in the reduction. In Lemma 3.3, we show that a type II adversary can be used to break RSA with a loss of a t factor in the reduction, where t is a polynomial-size "bound on $2^{\lceil \lg(x) \rceil}$" set to $4\ell[q+\lambda]$. The value t is established to avoid a circularity. In the proof of Lemma 3.3, the simulator would like to guess the index of the adversary's forgery in the range 1 to $2^{\lceil \lg(x) \rceil}$, so that it can set the function H_K accordingly. However, the simulator cannot compute x until it sets H_K. To avoid this circularity, we bound $t \geq 2^{\lceil \lg(x) \rceil}$. We want that for a random function R, there are at least q primes in the set of $\{R(i)\}_{i \in [1,x]}$ with high probability. Lemma B.1 in Appendix B guarantees that this occurs for $x = 2\ell[q+\lambda]$ with probability $1 - e^{-[q+\lambda](\frac{1}{2})^2}$. Thus, we set $t = 2x = 4\ell[q+\lambda]$, which establishes that $t \geq 2^{\lceil \lg(x) \rceil}$. This concludes the proof. □

3.2.1 Type I Adversary

Lemma 3.2. *If a type I adversary succeeds with probability ϵ, then it can be used to solve factoring with probability $\epsilon/(2\lambda) - negl(\lambda)$.*

Proof. We provide a reduction showing how to turn a type I adversary into an simulated adversary against factoring. Intuitively, in the simulation, the simulator takes a guess of $k^* = \lceil \lg(i) \rceil$, the logarithm of the index i on which the type I adversary will forge. There are at most λ values of k^* so the simulator has at least a $1/\lambda$ chance of guessing correctly.

We now describe the simulation. Given a factoring challenge $N = p_1 p_2$, where the goal is to produce either p_1 or p_2, proceed as follows.

Setup The simulator begins by guessing a value k^* in the range 1 to λ. Next, the simulator selects a random PRF key K and random $c \in \{0,1\}^\ell$, which defines the hash function $H_K()$. Next, for $i = 1$ to t, the simulator computes $e_i = H_K(i)$ and tests to see if it is prime. If e_i is prime, the

simulator places i into a set E. If $|E| < q$, the simulator aborts. In Lemma B.1, we show that due to our choice of t there will be at least q primes in E with high probability.

The simulator randomly chooses parameters L for a chameleon hash function, where $\{0,1\}^{\frac{2\ell}{3}}$ is the range of the hash. Finally, it chooses a random $u', h' \in \mathbb{Z}_N^*$ and sets

$$\hat{u} = (u')^{\prod_{j \in E} e_j} \quad \text{and} \quad \hat{h} = (h')^{\prod_{j \in E} e_j}$$
$$u = (\hat{u})^{2^{k^*}} \quad \text{and} \quad h = (\hat{h})^{2^{k^*}}.$$

Since u', h' are independently chosen, this will have the correct distribution.

The simulator outputs the public key as (N, u, h, c, K, L), sets the internal signing state $s = 0$ and keeps secret the chameleon hash trapdoor.

Sign When the adversary asks for a signature on message M, the simulator first updates its state value $s = s + 1$. Since the adversary is polynomial, we know that $s < 2^\lambda$. If $\lceil \lg(s) \rceil \geq k^*$, the simulator's guess was incorrect and it aborts. Otherwise, the simulator selects a random r, computes $x = \text{ChamHash}(M, r)$ and outputs the signature as:

$$\sigma_1 = \left((u'^x h') \prod_{j \in E}^{j \neq s} e_j \right)^{2^{(k^* - \lceil \lg(s) \rceil)}}, \quad r, \quad s.$$

Response Eventually, the type I adversary outputs a valid signature $\tilde{\sigma} = (\tilde{\sigma_1}, \tilde{r}, \tilde{i})$ on a message $\tilde{M}$. If $k^* \neq \lceil \lg(\tilde{i}) \rceil$, the simulator aborts. Otherwise, the simulator computes $x = \text{ChamHash}(\tilde{M}, \tilde{r})$. From the verification equation and simulation setup, we see that

$$(\tilde{\sigma_1}^{e^*})^{2^{k^*}} = u^x h = (\hat{u}^x \hat{h})^{2^{k^*}}.$$

Since N is a Blum-Williams integer, it follows that $(\tilde{\sigma_1^{e^*}})^2 = (\hat{u}^x \hat{h})^2$. Futhermore, the fact that h' was chosen randomly in $\mathbb{Z}_N^*$ and h' is raised to a product of odd primes e_i implies that with probability $\frac{1}{2}$ the value $\tilde{\sigma_1}^{e^*}$ is congruent to $\hat{u}^x \hat{h}$ modulo p_1, but not congruent modulo p_2 or vice versa. In this case, the simulator can factor N in the standard way, i.e., by computing a factor as $\mathbf{gcd}(\tilde{\sigma_1}^{e^*} - \hat{u}^x \hat{h}, N)$. □

3.2.2 Type II Adversary

Lemma 3.3. *If a type II adversary succeeds with probability ϵ after making q signing queries, then it can be used to solve RSA where N is a Blum-Williams integer with probability $\epsilon/(4\ell[q+\lambda]) - negl(\lambda)$.*

Proof. We provide a reduction showing how to turn a type II adversary into an adversary against RSA. Our proof has two components. First, we'll describe a simulator and show that any adversary which is successful against the simulation can be used to break RSA. Second, we'll show that any adversary successful against the above signature scheme will also be successful against the simulation.

Intuitively, in the simulation, the simulator takes a guess of i^*, the index on which the type II adversary will forge, within a small range of 1 to t. We'll lose a factor of t here. The simulator will choose public parameters such that it is straightforward to sign for any index except i^*. If the simulator is asked to sign for index i^*, we program the Chameleon hash to allow this.

We now describe the simulation. Given an RSA challenge (N, e^*, y), where the goal is to produce a value $w \in \mathbb{Z}_N^*$ such that $w^{e^*} = y$, we proceed as follows. For notation, let $N = p_1 p_2$.

Setup The simulator begins by guessing an index i^* in the range 1 to t. Next, the simulator selects a random PRF key K. It computes $c = F_K(i^*) \oplus e^*$, which defines the hash function $H_K()$. Recall that $e^* < 2^\ell$ since we "threw out" any other RSA challenge. Next, for $i = 1$ to t, the simulator computes $e_i = H_K(i)$ and tests to see if it is prime. If e_i is prime, the simulator places i into a set E. If $|E| < q$, the simulator aborts. In Lemma B.1, we show that due to our choice of t there will be at least q primes in E with high probability.

The simulator randomly chooses parameters L for a chameleon hash function, where $\{0,1\}^{\frac{2\ell}{3}}$ is the range of the hash. The simulator then selects a random value $x^* \in \{0,1\}^{\frac{2\ell}{3}}$.

Finally, it chooses a random $d \in \mathbb{Z}_N^*$ and sets

$$\begin{array}{rcrcl} \hat{u} = y^{\prod_{j\in E}^{j\neq i^*} e_j} & & \text{and} & & \hat{h} = \hat{u}^{-x^*} d^{\prod_{j\in E} e_j}, \text{ and then} \\ u = (\hat{u})^{2^\lambda} & & \text{and} & & h = (\hat{h})^{2^\lambda}. \end{array}$$

Since y, d are independently chosen, this will have the correct distribution.

The simulator outputs the public key as (N, u, h, c, K, L), sets the internal signing state $s = 0$ and keeps secret the chameleon hash trapdoor.

Sign When the adversary asks for a signature on message M, the simulator first updates its state value $s = s + 1$. Clearly, $s < 2^\lambda$. Now, there are two cases for computing the signature.

If $s = i^*$, then the simulator will employ the chameleon hash trapdoor to find a value r such that $\text{ChamHash}(M, r) = x^*$. The simulator then outputs the signature:

$$\sigma_1 = (d^{\prod_{j\in E}^{j\neq i^*} e_j})^{2^{(\lambda - \lceil \lg(i^*)\rceil)}}, \quad r, \quad i^*.$$

To verify correctness, notice that we can rewrite σ_1 as follows:

$$\begin{array}{rcl} \sigma_1 & = & (\hat{u}^{x^*/e^*} \cdot \hat{u}^{-x^*/e^*} \cdot d^{\prod_{j\in E}^{j\neq i^*} e_j})^{2^{(\lambda - \lceil \lg(i^*)\rceil)}} \\ & = & (\hat{u}^{x^*/e^*} \cdot \hat{h}^{1/e^*})^{2^{(\lambda - \lceil \lg(i^*)\rceil)}} \\ & = & \left((\hat{u}^{x^*}\hat{h})^{1/e^*}\right)^{2^{(\lambda - \lceil \lg(i^*)\rceil)}} = ((u^{x^*}h)^{1/e*})^{(\frac{1}{2})^{\lceil \lg(i^*)\rceil}} \end{array}$$

If $s \neq i^*$, then the simulator chooses a random r and computes $x = \text{ChamHash}(M, r)$. The simulator then outputs the signature:

$$\sigma_1 = \left((y^{x\prod_{j\in E}^{j\neq s, j\neq i^*} e_j}) \cdot (y^{-x^*\prod_{j\in E}^{j\neq s, j\neq i^*} e_j}) \cdot (d^{\prod_{j\in E}^{j\neq s} e_j})\right)^{2^{(\lambda - \lceil \lg(s)\rceil)}}, \quad r, \quad s.$$

To verify correctness, notice that we can rewrite σ_1 as follows:

$$\sigma_1 = ((\hat{u}^x\hat{h})^{1/e_s})^{2^{(\lambda - \lceil \lg(s)\rceil)}} = ((u^x h)^{1/e_s})^{(\frac{1}{2})^{\lceil \lg(s)\rceil}}$$

Response Eventually, the type II adversary outputs a valid signature $\tilde{\sigma} = (\tilde{\sigma_1}, \tilde{r}, \tilde{i})$ on a message $\tilde{M}$ such that $\tilde{i} \leq t$. If $\tilde{i} \neq i^*$, the simulator's guess was incorrect and it aborts.

Otherwise, the simulator computes $x = \text{ChamHash}(\tilde{M}, \tilde{r})$. If $x = x^*$, the simulator aborts. Otherwise, from the verification equation and simulation setup, we see that

$$(\tilde{\sigma_1}^{e^*})^{2^{\lceil \lg(\tilde{i})\rceil}} = u^x h = (\hat{u}^x\hat{h})^{2^\lambda}$$

We can see that $(\tilde{\sigma_1}^{e^*})^2 = (\hat{u}^x\hat{h})^{2^{\lambda-\lceil\lg(\tilde{i})\rceil+1}}$ via Lemma 2.3 since $\lambda > \lceil\lg(\tilde{i})\rceil$ and N is a Blum-Williams integer. Thus, we have two cases to consider regarding the underlying square roots.

- Case A: $\tilde{\sigma_1}^{e^*}$ or $-\tilde{\sigma_1}^{e^*}$ is equal to $(\hat{u}^x\hat{h})^{2^{\lambda-\lceil\lg(\tilde{i})\rceil}} \bmod N$.
- Case B: $\tilde{\sigma_1}^{e^*}$ is congruent to $(\hat{u}^x\hat{h})^{2^{\lambda-\lceil\lg(\tilde{i})\rceil}} \bmod p_1$, but not congruent mod p_2 or vice versa.

Case A: Suppose $v = \tilde{\sigma_1}^{e^*}$ is congruent to $(\hat{u}^x\hat{h})^{2^{\lambda-\lceil\lg(\tilde{i})\rceil}} \bmod N$. (The case for $-v$ is analogous.) Clearly, v is a square modulo N. Since $\lambda > \lceil\lg(\tilde{i})\rceil$, we can apply Lemma 2.3 to obtain

$$\tilde{\sigma_1}^{e^*} = (\hat{u}^x\hat{h})^{2^{\lambda-\lceil\lg(\tilde{i})\rceil}}.$$

Let $v = \sigma_1/(d^{\prod_{j\in E}^{j\neq i^*} e_j})^{2^{\lambda-\lceil\lg(\tilde{i})\rceil}}$. Then substituting into the above equation we have:

$$v^{e^*} = y^{2^{\lambda-\lceil\lg(\tilde{i})\rceil}(x-x^*)\prod_{j\in E}^{j\neq i^*} e_j}.$$

The simulator now runs the algorithm from Lemma 2.2 to obtain a value w such that $w^{e^*} = y$, and outputs w as the solution to the RSA challenge. We may apply Lemma 2.2 since (1) $w, y \in \mathbb{Z}_N$, (2) e^* and $2^{\lambda-\lceil\lg(\tilde{i})\rceil}(x-x^*)\prod_{j\in E}^{j\neq i^*} e_j$ are in $\mathbb{Z}$, and (3) e^* is relatively prime to $2^{\lambda-\lceil\lg(\tilde{i})\rceil}(x-x^*)\prod_{j\in E}^{j\neq i^*} e_j$ with high probability as shown in Claim B.2 of Appendix B.

Case B: The simulator computes $\mathbf{gcd}(\tilde{\sigma_1}^{e^*} - (\hat{u}^x\hat{h})^{2^{\lambda-\lceil\lg(\tilde{i})\rceil}}, N)$ to obtain a factor of N.

This ends our description of the simulator.

We now argue that any successful type II adversary against our scheme will have success in the game presented by the simulator. To do this, we first define a sequence of games, where the first game models the real world and the final game is exactly the view of the adversary when interacting with our simulator. We then show via a series of claims that if a type II adversary is successful against Game j, then it will also be successful against Game $j+1$.

Game 1: This game is defined to be the same as the security game of the scheme.

Game 2: The challenger chooses a random value i^* between 1 and t, and $y^* \in \{0,1\}^\ell$. It chooses K randomly, but sets $c = y^* \oplus F_K(i^*)$. Furthermore, the adversary is only considered successful if she produces a forgery on index i^* (i.e., all forgeries not on i^* are not considered successful in this game.)

Game 3: The same as Game 2, with the following exception. Consider the i^*-th signature (σ_1, r, i^*) on message M. Let $x^* = \text{ChamHash}(M, r)$. In addition to the constraints above, the adversary will also not be considered successful if she produces a forgery $(\tilde{\sigma_1}, \tilde{r}, \tilde{i})$ on message $\tilde{M}$ such that $x^* = \text{ChamHash}(\tilde{M}, \tilde{r})$.

Game 4: The same as Game 3, with the exception that y^* is chosen to be a random odd *prime* in the range $\{0,1\}^\ell$.

Notice that Game 4 is exactly the view of the adversary when interacting with our simulator. We now establish a series of claims that show that if a type II adversary is successful against the real security game (Game 1) then it will be successful against our RSA simulation (Game 4).

Define $\mathbf{Adv}_{\mathcal{A}}[\text{Game } x]$ as the advantage of a type II adversary $\mathcal{A}$ in Game x.

Claim 3.4. *If F is a secure pseudorandom function, then*

$$\mathbf{Adv}_{\mathcal{A}}[\text{Game 2}] = \frac{\mathbf{Adv}_{\mathcal{A}}[\text{Game 1}]}{t} - negl(\lambda).$$

Proof. By our definition of y^*, c is chosen with the same distribution as in Game 1. What we now want to establish is that the highest index x of a signature that $\mathcal{A}$ sees is less than or equal to t. Consider a truly random function R. The probability for any given i that $R(i) \oplus c$ is a prime is greater than or equal to $\frac{1}{\ln(2^\ell)} \geq \frac{1}{\ell}$ due to Theorem 2.4 and is independent. It follows from our definition of t and Chernoff bounds that the number of $1 \leq i \leq t$ such that $R(i) \oplus c$ is prime is less than q (actually $q + \lambda$) happens with negligible probability $e^{-[q-\lambda](\frac{1}{2})^2}$. If, for a random key K, the PRF F_K produced less than q primes with non-negligible probability, then this would admit a distinguishing attack (which again does *not* depend on the adversary's possession of K.) □

Claim 3.5. *If* ChamHash *is a secure chameleon hash function, then*

$$\mathbf{Adv}_{\mathcal{A}}[\text{Game 3}] = \mathbf{Adv}_{\mathcal{A}}[\text{Game 2}] - negl(\lambda).$$

Proof. An adversary $\mathcal{A}$ succeeds in Game 3 whenever it succeeds in Game 2, except the case where $\mathcal{A}$ manages to find a collision in the chameleon hash. That is, whenever the simulator provided values (M, r) such that $\text{ChamHash}(M, r) = x^*$ and then $\mathcal{A}$ was able to produce values $(\tilde{M}, \tilde{r})$, where $M \neq \tilde{M}$, and yet $\text{ChamHash}(\tilde{M}, \tilde{r}) = x^*$. By definition, no polynomial-time adversary can find collisions in a secure chameleon hash with better than negligible probability. □

Claim 3.6. *It holds that* $\mathbf{Adv}_{\mathcal{A}}[\text{Game 4}] \geq \mathbf{Adv}_{\mathcal{A}}[\text{Game 3}]$.

Proof. For any adversary to be successful in Game 3, it is necessary that $H_K(i^*) = y^*$ is a prime; this follows simply from the fact that any verifier will reject a signature on any index that does not evaluate to a prime. Let $\mathbf{Win}_{\mathcal{A}}$ denote the event that adversary $\mathcal{A}$ is successful in a forgery.

$$\begin{aligned}
\Pr[\mathbf{Win}_{\mathcal{A}} \text{ in Game 3}] &= \sum_{1 \leq z \leq 2^\ell} \frac{\Pr[\mathbf{Win}_{\mathcal{A}} | y^* = z]}{2^\ell} = \sum_{1 \leq z \leq 2^\ell} \frac{\Pr[\mathbf{Win}_{\mathcal{A}} | y^* = z \wedge z \text{ is prime}]}{2^\ell} \\
\Pr[\mathbf{Win}_{\mathcal{A}} \text{ in Game 4}] &= \sum_{1 \leq z \leq 2^\ell, z \text{ is prime}} \frac{\Pr[\mathbf{Win}_{\mathcal{A}} | y^* = z]}{\ell}
\end{aligned}$$

And thus, we see that $\Pr[\mathbf{Win}_{\mathcal{A}} \text{ in Game 3}] \leq \Pr[\mathbf{Win}_{\mathcal{A}} \text{ in Game 4}]$. □

□

4 Our CDH Realization

Our CDH construction is both simplier and more efficient than its RSA counterpart. This is partly due to the fact that here will not need to search for primes and can instead directly associate the ith state with the ith signature. Here we will also directly embed the chameleon hash function.

As before, each signature is associated with an index i and a category $k = \lceil \lg(i) \rceil$. We force the adversary to forge on a previously seen category, which restricts her to a polynomial-size set from which to choose her forgery index. Since this remaining set is polynomial in size, we can employ selectively-secure techniques to obtain an adaptively-secure scheme. Specifically, we make use of the selectively-secure signatures due to Boneh and Boyen [5] with a twist. Here our index is like the message in their scheme and our message impacts their "master key".

4.1 CDH Construction

Setup(1^λ) The setup algorithm selects a bilinear group $\mathbb{G}$ of prime order $p > 2^\lambda$. It chooses a random exponent $a \in \mathbb{Z}_p$. It chooses random group elements $g, u, v, d, w, z, h \in \mathbb{G}$. The public key is output as:

$$g, g^a, u, v, d, w, z, h.$$

The setup algorithm sets its state counter $s = 0$ and keeps a as the secret key SK.

Sign$(\text{SK}, s, M \in \mathbb{Z}_p)$ The message space is treated as $\mathbb{Z}_p$; to sign arbitrarily long messages one could first apply a collision-resistant hash function. The signer first increments its counter s by one as $s = s + 1$. If $s > 2^\lambda$, then abort. Otherwise, the algorithm chooses random $r, t \in \mathbb{Z}_p$ and then outputs the signature as:

$$\sigma_1 = (u^M v^r d)^a (w^{\lceil lg(s) \rceil} z^s h)^t, \quad \sigma_2 = g^t, \quad r, \quad s.$$

Conceptually, we can think of t as being the randomness from the Boneh-Boyen selectively-secure signature [5] and r as being the randomness for a Chameleon hash function $u^M v^r$.

Verify$(\text{PK}, M, \sigma = (\sigma_1, \sigma_2, r, i))$ The verification algorithm first makes sure that $i < 2^\lambda$. If it is greater, then it rejects. Then it uses the bilinear map to verify the signature by checking that

$$e(\sigma_1, g) = e(u^M v^r d, g^a) e(\sigma_2, w^{\lceil \lg(i) \rceil} z^i h).$$

4.2 Comments

We first note that the verification cost can be reduced to only two pairings by publishing or having each verifier do a one-time precomputation of the values $e(u, g^a), e(v, g^a)$ and $e(d, g^a)$. This is competitive with the most efficient bilinear schemes in the random oracle model, e.g., [7].

Second, we assume the adversary never makes more than 2^λ signature requests and that we choose $p > 2^\lambda$. If somehow the adversary did make more that 2^λ signature requests, then we could simply give the adversary the secret key. This holds for both the CDH and RSA schemes.

Finally, we embedded a specific Chameleon hash function into our CDH scheme, because this appears to be the most efficient construction. We could, however, have set the signature as:

$$\sigma_1 = (u^x d)^a (w^{\lceil lg(s) \rceil} z^s h)^t, \quad \sigma_2 = g^t, \quad r, \quad s.$$

where $x = \text{ChamHash}(M, r)$ for *any* chameleon hash function mapping into $\mathbb{Z}_p$. However, the public parameters necessary for the chameleon hash would likely eclipse the gains of removing element v.

4.3 Proof of Security

Theorem 4.1. *If the CDH assumption holds in $\mathbb{G}$, then the above construction is a secure signature scheme.*

Proof. Suppose we have an adversary that makes at most $q(\lambda)$ queries where $q()$ is a polynomial. (We say q queries where it is clear from context.) We show that this adversary breaks CDH. We do so by noting an adversary can have two types of forgeries.

Type I The adversary forges for a message with index i greater than $2^{\lceil \lg(q) \rceil}$.

Type II The adversary forges for a message with index i less than or equal to $2^{\lceil \lg(q) \rceil}$.

In Lemma 4.2, we show that a type I adversary can be used to break CDH with a loss of a λ factor in the reduction. In Lemma 4.3, we show that a type II adversary can be used to break CDH with a loss of a q factor in the reduction, where q is the number of signing queries made by the adversary. This concludes the proof. □

4.3.1 Type I Adversary

Lemma 4.2. *If a type I adversary succeeds with probability ϵ, then it can be used to solve CDH with probability ϵ/λ.*

Proof. A type I adversary is caught by taking a guess of $k^* = \lceil \lg(i) \rceil$, the logarithm of the index i on which he will forge. There are at most λ values of k^* so the simulator has at least a $1/\lambda$ chance of guessing correctly. It then uses the selective Boneh-Boyen simulation [5] to answer all other queries. Let us now explain the details. Given a CDH challenge (g, g^a, g^b), proceed as follows.

Setup The simulator begins by guessing a value k^* in the range 1 to λ. This represents a guess that the adversary will forge on index i such that $k^* = \lceil \lg(i) \rceil$. For type I adversaries, recall that if the adversary forges using value k^*, it will not ask enough signing queries to see an original signature using k^*. Recall that the verification algorithm rejects on all indexes greater than 2^λ.

Next, choose random $y_u, y_v, y_z \in \mathbb{Z}_p$ and set $u = g^{y_u}, v = g^{y_v}, z = g^{y_z}$. (These values will play no significant role outside of realizing the proper distribution.) Then set $d = g^b$, $w = g^b g^{x_w}$, and $h = g^{-bk^*} g^{x_h}$, for random $x_w, x_h \in \mathbb{Z}_p$.

The simulator outputs the public key as $(g, g^a, u, v, d, w, z, h)$, sets the internal signing state $s = 0$, and implicitly designates the secret key as a.

Sign When the adversary asks for a signature on message $M \in \mathbb{Z}_p$, the simulator first updates its state value $s = s + 1$. If $s > 2^\lambda$, the simulator would need to abort. Fortunately, this never occurs since, by definition, a polynomial-time adversary would make less than 2^λ. If $k^* = \lceil \lg(s) \rceil$, the simulator aborts. Otherwise, it computes the signature by choosing random $r, t' \in \mathbb{Z}_p$, computing $k = \lceil \lg(s) \rceil$ and $T = g^{t'}/(g^a)^{1/(k-k^*)} = g^{t' - a/(k-k^*)}$, and outputting:

$$\sigma_1 = (g^a)^{y_u M} \cdot (g^a)^{y_v r} \cdot T^{x_w k + y_z s + x_h} \cdot (g^b)^{t'(k-k^*)} \quad , \quad \sigma_2 = T \quad , \quad r \quad , \quad s.$$

Let us implicitly set the randomness $t = t' - a/(k - k^*)$ (here t' gives t the proper distribution) and we have $T = g^t$ and

$$\sigma_1 = (u^M v^r)^a \cdot (g^{x_w k} z^s g^{x_h})^t \cdot g^{bt'(k-k^*)} \quad , \quad \sigma_2 = g^t \quad , \quad r \quad , \quad s.$$

To verify correctness, notice that we can rewrite σ_1 as follows:

$$\begin{aligned}
\sigma_1 &= (u^M v^r)^a \cdot g^{ab} \cdot (g^{x_w k} z^s g^{x_h})^t \cdot g^{bt'(k-k^*)} \cdot g^{-ab} \\
&= (u^M v^r)^a \cdot g^{ab} \cdot (g^{x_w k} z^s g^{x_h})^t \cdot g^{bt'(k-k^*)} \cdot g^{-ab(k-k^*)/(k-k^*)} \\
&= (u^M v^r)^a \cdot g^{ab} \cdot (g^{x_w k} z^s g^{x_h})^t \cdot g^{bk(t'-a/(k-k^*))} \cdot g^{-bk^*(t'-a/(k-k^*))} \\
&= (u^M v^r)^a \cdot g^{ab} \cdot (g^{x_w k} z^s g^{x_h})^t \cdot g^{bkt} \cdot g^{-bk^*t} \\
&= (u^M v^r g^b)^a ((g^b g^{x_w})^k z^s (g^{-bk^*} g^{x_h}))^t \\
&= (u^M v^r h)^a (w^k z^s h)^t
\end{aligned}$$

Response Eventually, the type I adversary outputs a valid signature $\tilde{\sigma} = (\tilde{\sigma_1}, \tilde{\sigma_2}, \tilde{r}, \tilde{i})$ on a message $\tilde{M} \in \mathbb{Z}_p$ such that $\tilde{i} \geq 2q$. From the verification equation, we see that

$$e(\tilde{\sigma_1}, g) = e(u^{\tilde{M}} v^{\tilde{r}} d, g^a) e(\tilde{\sigma_2}, w^{\lceil \lg(\tilde{i}) \rceil} z^{\tilde{i}} h).$$

Interpreting $\tilde{\sigma_2}$ as g^t, for some $t \in \mathbb{Z}_p$, it follows from the above equation that

$$\tilde{\sigma_1} = (u^{\tilde{M}} v^{\tilde{r}} d)^a (w^{\lceil \lg(\tilde{i}) \rceil} z^{\tilde{i}} h)^t.$$

Let $\tilde{k} = \lceil \lg(\tilde{i}) \rceil$. If $k^* = \tilde{k}$, then the simulator guessed correctly and we know that

$$\begin{aligned}
\tilde{\sigma_1} &= (u^{\tilde{M}} v^{\tilde{r}} d)^a ((g^{b+x_w})^{\tilde{k}} (g^{y_z})^{\tilde{i}} (g^{-bk^*+x_h}))^t \\
&= (u^{\tilde{M}} v^{\tilde{r}} d)^a (g^{x_w \tilde{k}} g^{y_z \tilde{i}} g^{x_h})^t \\
&= (g^{\tilde{M} y_u} g^{\tilde{r} y_v} g^b)^a (g^{x_w \tilde{k}} g^{y_z \tilde{i}} g^{x_h})^t \\
&= g^{ab} (g^{\tilde{M} y_u} g^{\tilde{r} y_v})^a (g^{x_w \tilde{k}} g^{y_z \tilde{i}} g^{x_h})^t \\
&= g^{ab} (g^a)^{\tilde{M} y_u + \tilde{r} y_v} (g^t)^{x_w \tilde{k} + y_z \tilde{i} + x_h}
\end{aligned}$$

Thus, the simulator outputs $\tilde{\sigma_1} / ((g^a)^{\tilde{M} y_u + \tilde{r} y_v} (g^t)^{x_w \tilde{k} + y_z \tilde{i} + x_h}) = g^{ab}$. If $k^* \neq \tilde{k}$, the simulator aborts. □

4.3.2 Type II Adversary

Lemma 4.3. *If a type II adversary succeeds with probability ϵ after making q signing queries, then it can be used to solve CDH with probability $\epsilon / O(q)$.*

Proof. A type II adversary is caught by taking a guess of i^*, the index on which he will forge, within a small range of 1 to $2^{\lceil \lg(q) \rceil}$, where q is the number of signing queries made by the adversary. Note we loose a q factor in our reduction with this guess. At a high-level, we will again use the selective Boneh-Boyen simulation [5] to sign for all indexes $i \neq i^*$. When signing for the i^*-th signature, we program the Chameleon hash to allow us to do this. Let us now explain the details. Given a CDH challenge (g, g^a, g^b), proceed as follows.

Setup The simulator begins by guessing an index i^* in the range 1 to $2^{\lceil \lg(q) \rceil}$. This represents a guess that the adversary will choose to forge on index i^*.

Next, choose random $y_v, y_d, x_v, x_d \in \mathbb{Z}_p$ and set $u = g^b$, $v = g^{b x_v} g^{y_v}$ and $d = g^{b x_d} g^{y_d}$. Then choose random $y_w, x_z, x_h \in \mathbb{Z}_p$ and set $w = g^{y_w}$, $z = g^b g^{x_z}$ and $h = g^{-b i^*} g^{x_h}$.

The simulator outputs the public key as $(g, g^a, u, v, d, w, z, h)$, sets the internal signing state $s = 0$, and implicitly designates the secret key as a.

Sign When the adversary asks for a signature on message $M \in \mathbb{Z}_p$, the simulator first updates its state value $s = s + 1$. There are now two ways the simulator will proceed.

If $s = i^*$, then program the chameleon hash function by first computing $r = (x_d - M)/x_v$. Then choose random $t \in \mathbb{Z}_p$ and set

$$\sigma_1 = (g^a)^{y_v r + y_d} \cdot (w^{\lceil \lg(s) \rceil} z^s h)^t \quad , \quad \sigma_2 = g^t \quad , \quad r \quad , \quad s.$$

To verify correctness, observe that we can rewrite σ_1 as follows given that $M + r x_v = x_d$:

$$\begin{aligned} \sigma_1 &= (g^{ab})^{M + r x_v - x_d} (g^a)^{y_v r + y_d} \cdot (w^{\lceil \lg(s) \rceil} z^s h)^t \\ &= (g^{bM} g^{(b x_v + y_v) r} g^{-b x_d + y_d})^a \cdot (w^{\lceil \lg(s) \rceil} z^s h)^t \\ &= (u^M v^r d)^a \cdot (w^{\lceil \lg(s) \rceil} z^s h)^t \end{aligned}$$

If $s \neq i^*$, then we follow the Boneh-Boyen simulation by choosing random $r, t' \in \mathbb{Z}_p$, computing $T = g^{t'} / (g^a)^{(M + x_v r + x_d)/(s - i^*)} = g^{t' - a(M + x_v r + x_d)/(s - i^*)}$, and outputting:

$$\sigma_1 = (g^a)^{y_v r + y_d} \cdot T^{y_w \lceil \lg(s) \rceil + x_z s + x_h} \cdot (g^b)^{t'(s - i^*)} \quad , \quad \sigma_2 = T \quad , \quad r \quad , \quad s.$$

Let us implicitly set the randomness $t = t' - a(M + x_v r + x_d)/(s - i^*)$ (here t' gives t the proper distribution) and we have $T = g^t$ and

$$\sigma_1 = (g^{y_v r} g^{y_d})^a \cdot (w^{\lceil \lg(s) \rceil} g^{x_z s} g^{x_h})^t \cdot (g^b)^{t'(s - i^*)} \quad , \quad \sigma_2 = g^t \quad , \quad r \quad , \quad s.$$

To verify correctness, notice that we can rewrite σ_1 as follows:

$$\begin{aligned} \sigma_1 &= (g^{ab})^{(M + x_v r + x_d)} (g^{y_v r} g^{y_d})^a \cdot (w^{\lceil \lg(s) \rceil} g^{x_z s} g^{x_h})^t \cdot (g^b)^{t'(s - i^*)} (g^{-ab})^{(M + x_v r + x_d)} \\ &= ((g^b)^M (g^{b x_v + y_v})^r (g^{b x_d + y_d}))^a \cdot (w^{\lceil \lg(s) \rceil} g^{x_z s} g^{x_h})^t \cdot (g^b)^{t'(s - i^*)} (g^{-ab})^{(M + x_v r + x_d)} \\ &= (u^M v^r d)^a \cdot (w^{\lceil \lg(s) \rceil} g^{x_z s} g^{x_h})^t \cdot (g^b)^{t'(s - i^*)} (g^{-ab})^{(M + x_v r + x_d)} \\ &= (u^M v^r d)^a \cdot (w^{\lceil \lg(s) \rceil} g^{x_z s} g^{x_h})^t \cdot (g^{b(s - i^*)})^t \\ &= (u^M v^r d)^a \cdot (w^{\lceil \lg(s) \rceil} (g^{b + x_z})^s g^{-b i^* + x_h})^t \\ &= (u^M v^r d)^a \cdot (w^{\lceil \lg(s) \rceil} z^s h)^t \end{aligned}$$

Response Eventually, the type II adversary outputs a valid signature $\tilde{\sigma} = (\tilde{\sigma_1}, \tilde{\sigma_2}, \tilde{r}, \tilde{i})$ on a message $\tilde{M} \in \mathbb{Z}_p$ such that $\tilde{i} < 2^{\lceil \lg(q) \rceil}$. Let $\tilde{\sigma_2} = g^t$ for some $t \in \mathbb{Z}_p$. Now, from the verification equation, we see that

$$\tilde{\sigma_1} = (u^{\tilde{M}} v^{\tilde{r}} d)^a (w^{\lceil \lg(\tilde{i}) \rceil} z^{\tilde{i}} h)^t.$$

If $i^* = \tilde{i}$, then the simulator guessed correctly and we know that

$$\begin{aligned} \tilde{\sigma_1} &= ((g^b)^{\tilde{M}} (g^{b x_v + y_v})^{\tilde{r}} (g^{b x_d + y_d}))^a ((g^{y_w})^{\lceil \lg(\tilde{i}) \rceil} (g^{b + x_z})^{\tilde{i}} (g^{-b i^* + x_h}))^t \\ &= g^{ab(\tilde{M} + x_v \tilde{r} + x_d)} g^{a(y_v \tilde{r} + y_d)} ((g^{y_w})^{\lceil \lg(\tilde{i}) \rceil} (g^{b + x_z})^{\tilde{i}} (g^{-b i^* + x_h}))^t \\ &= g^{ab(\tilde{M} + x_v \tilde{r} + x_d)} g^{a(y_v \tilde{r} + y_d)} (g^{y_w \lceil \lg(\tilde{i}) \rceil} g^{x_z \tilde{i}} g^{x_h})^t \\ &= g^{ab(\tilde{M} + x_v \tilde{r} + x_d)} g^{a(y_v \tilde{r} + y_d)} g^{t(y_w \lceil \lg(\tilde{i}) \rceil + x_z \tilde{i} + x_h)} \end{aligned}$$

If $(\tilde{M} + x_v \tilde{r} + x_d) \neq 0$, the simulator outputs $(\tilde{\sigma_1} / (g^{a(y_v \tilde{r} + y_d)} \tilde{\sigma_2}^{(y_w \lceil \lg(\tilde{i}) \rceil + x_z \tilde{i} + x_h)}))^{1/(\tilde{M} + x_v \tilde{r} + x_d)} = g^{ab}$. Otherwise, the simulator aborts. The probability that $(\tilde{M} + x_v \tilde{r} + x_d) = 0$ is $1/p$. To see this,

observe that the values x_v and x_d are initially hidden by blinding factors y_v and y_d, respectively. For all $i \neq i^*$, signatures are produced without regard to these values and contain no information about either x_v or x_d. For the one signature where $i = i^*$, the adversary could obtain the information that $M_i + x_v r_i + x_d = 0$. However, there are exactly p possible (x_v, x_d) pairs that satisfy this equation and each of them are equally likely. Thus, information-theoretically, the adversary can output a pair $(\tilde{M}, \tilde{r})$ satisfying $\tilde{M} + x_v \tilde{r} + x_d = 0$ with probability at most $1/p$. □

5 Handling State in Practice

One of the challenging issues when using our signature scheme in practice is that a signer must maintain state. Issues that may arise in practice include multiple (autonomous) machines sharing the same signing key and machine crashes, among other problems. Fortunately, in our scheme, since the state is a simple counter, most of these issues can be readily addressed.

Multiple Signers. For a variety of reasons, administrators often set up multiple machines with the same signing key (e.g., parallelizing SSL connections at a highly visited site). In most cases, it is impractical to assume that all of the machines can coordinate to maintain a shared state. However, in our system, there is a simple solution to deal with this problem. If n different machines are using the same signing key, then machine i can give its jth signature with index $n \cdot j + i$.

Handling Machine Crashes. On an actual implementation, it is important to commit the counter increment (to persistent memory) before giving out the signature. Otherwise, a crash might cause two signatures to be given out for the same counter and thereby compromise security. We observe that it is perfectly fine to skip over a small (i.e., polynomial) number of counters and recommend erring on the side of safety.

Using the Machine Clock as a State. Instead of having a signer maintain a state counter, one interesting alternative is to use the machine clock time as the signer's state. This can theoretically work in our system since the clock time monotonically increases at a polynomial rate. One concern, however, is that the signer should not issue more than one signature per clock period. Two potential circumstances where this could arise are either if somehow the clock time was set backwards or the signing algorithm in two different invocations read the same clock value. This is especially relevant in the age of dual-core processors, where mechanisms are in place for managing access to shared memory, but there are not necessarily guarantees that two cores on the same chip would not read out the same clock value. Overall, using the clock as the state could be a risky design choice and a detailed analysis of the system implementation should be made before applying it.

6 Conclusion and Open Problems

We presented two practical hash-and-sign signatures based on RSA and CDH in bilinear groups in the standard model. We employed a new technique for restricting any adversary's ability to forge, which can be alternatively viewed as a mechanism for transforming selectively-secure techniques into adaptively-secure constructions.

We view our stateful constructions here as a step toward realizing similar *stateless* signatures. Recall that early tree-based signatures (e.g., the GMR signatures [21]) had a stateful signer, until

Goldreich [20] showed how to remove the state. Goldreich's techniques do not appear to apply directly here, but we are optimistic that similar progress can be made.

While we focused on RSA and CDH based constructions, it would also be interesting to realize constructions under CDH in a non-bilinear group, lattices, or general assumptions.

Finally, we note that hash-and-sign signatures and their extensions with efficient protocols (e.g., [9, 10]) have been useful for integrating into larger systems, such as anonymous credentials and e-cash. With this new building block and some additional work, one might be able to base these larger systems on more standard assumptions.

Acknowledgments

We thank Dan Boneh for valuable discussions, including suggesting the second variant of our RSA scheme described in Section 3.1.1. We also thank Giuseppe Ateniese, Matthew Green and the anonymous reviewers for helpful comments.

References

[1] Giuseppe Ateniese and Breno de Medeiros. Identity-based chameleon hash and application. In *Financial Cryptography '04*, volume 3110, pages 164–180, 2004.

[2] Mihir Bellare and Silvio Micali. How to sign given any trapdoor function. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO '88*, volume 403, pages 200–215, 1990.

[3] Mihir Bellare and Sara Miner. A forward-secure digital signature scheme. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO '99*, volume 1666, pages 431–448, 1999.

[4] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS*, pages 62–73, 1993.

[5] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In *Advances in Cryptology – EUROCRYPT '04*, volume 3027, pages 223–238, 2004.

[6] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *Advances in Cryptology — EUROCRYPT '04*, volume 3027, pages 54–73, 2004.

[7] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.

[8] Joan Boyar, S.A. Kurtz, and Mark W. Krentel. A discrete logarithm implementation of perfect zero-knowledge blobs. *Journal of Cryptology*, 2(2):63–76, 1990.

[9] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *Security in Communication Networks '02*, volume 2576, pages 268–289, 2002.

[10] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology – CRYPTO '04*, volume 3152, pages 56–72, 2004.

[11] Ronald Cramer and Ivan Damgård. Secure signature schemes based on interactive protocols. In *Advances in Cryptology — CRYPTO '95*, pages 297–310, 1995.

[12] Ronald Cramer and Ivan Damgård. New generation of secure and practical RSA-based signatures. In *Advances in Cryptology — CRYPTO '96*, pages 173–185, 1996.

[13] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. *ACM Transactions on Information and System Security*, 3(3):161–185, 2000.

[14] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[15] Cynthia Dwork and Moni Naor. Universal one-way hash functions and their cryptographic applications. In *Symposium on the Theory of Computation*, pages 33–43, 1989.

[16] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology — CRYPTO '84*, volume 196, pages 10–18, 1984.

[17] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In *Advances in Cryptology — EUROCRYPT '99*, volume 1592, pages 123–139, 1999.

[18] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Symposium on the Theory of Computing*, pages 197–206, 2008.

[19] Eu-Jin Goh, Stanislaw Jarecki, Jonathan Katz, and Nan Wang. Efficient signature schemes with tight reductions to the Diffie-Hellman problems. *J. of Cryptology*, 20(4):493–514, 2007.

[20] Oded Goldreich. Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In *Advances in Cryptology – CRYPTO '86*, volume 263, pages 104–110, 1986.

[21] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2), 1988.

[22] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *Network and Distributed System Security Symposium*, 2000.

[23] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO '89*, volume 435, pages 218–238, 1989.

[24] Gary L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13:300–317, 1976.

[25] David Naccache. Secure and practical identity-based encryption, 2005. Cryptology ePrint Archive: Report 2005/369.

[26] Moni Naor and Moti Yung. An efficient existentially unforgeable signature scheme and its applications. In *Advances in Cryptology – CRYPTO '94*, volume 839, pages 234–246, 1994.

[27] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *Advances in Cryptology – CRYPTO '92*, volume 740, pages 31–53, 1992.

[28] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, volume 1070, pages 387–398, 1996.

[29] Michael O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12:128–138, 1980.

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

[31] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Symposium on the Theory of Computing*, pages 387–394. ACM, 1990.

[32] Claus P. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.

[33] Brent Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT '05*, volume 3494, pages 320–329, 2005.

A Chameleon Hash Function based on RSA

While the chameleon hash function based on the hardness of factoring due to Krawczyk and Rabin [22] would be sufficient to rest our Section 3 construction entirely on the difficulty of RSA, we now present a more efficient option.

This construction is due to Ateniese and de Mederios [1]. Their work actually presents an identity-based chameleon hash, which we simplify, since we will not require the identity-based property. To obtain the identity-based feature, the authors employed a signature scheme secure in the random oracle model and proved the security of their scheme within this context. For completeness, we provide a proof of the basic hash function under RSA in the standard model, but the credit here should go to the prior work.

Let ℓ be a security parameter. Let N be an RSA modulus such that $2^\ell < \phi(N) < 2^{\ell+2}$. Choose a random, positive $e \in \{0,1\}^\ell$ which is relatively prime to $\phi(N)$ and a random value $J \in \mathbb{Z}_N$. Set the public key as (N, e, J) and keep as the trapdoor the factorization of N as well as a value d such that $ed \equiv 1 \mod \phi(N)$.

The hash $H : \{0,1\}^{\frac{2\ell}{3}} \times \mathbb{Z}_N \rightarrow \mathbb{Z}_N$ takes two inputs and produces one output. The hash is computed as $H(m, r) = J^m r^e \mod N$. The holder of the trapdoor can compute a collision for any message m' by solving the following equation for r':

$$J^m r^e = J^{m'} r'^e \text{ as } r' = r(J^d)^{m-m'} \mod N.$$

We note that the choice of $\{0,1\}^{\frac{2\ell}{3}}$ is somewhat arbitrary. It could be optimized to any constant fraction of ℓ bits or any range $\{0,1\}^{\ell'}$ such that the probability that $e \notin \{0,1\}^{\ell'}$ is non-negligible.

Theorem A.1. *The above chameleon hash function is secure under the RSA assumption in the standard model.*

Proof. We provide a reduction showing how to turn any adversary against the above chameleon hash into a simulated adversary that breaks RSA. Let (N, e, J) be the challenge RSA instance. If $e \geq 2^\ell$, abort. This happens with probability at most $(2^{\ell+2} - 2^\ell)/2^{\ell+2} = 3/4$. Otherwise, the

simulator sets the chameleon hash public key as (N, e, J). Suppose the adversary can find a collision pair $(m, r), (m', r')$ such that $m \neq m'$ and yet $J^m r^e = J^{m'} r'^e$. We rewrite this as $(r'/r)^e = J^{m-m'}$. The simulator now wants to apply Lemma 2.2 to obtain a value z such that $z^e \equiv J \mod N$ and output z. However, this lemma only applies if e and $(m - m')$ are relatively prime. We now argue that this is the case with non-negligible probability. Recall that $e \in \{0,1\}^\ell$ and $(m-m') \in \{0,1\}^{\frac{2\ell}{3}}$. We first observe that $e = (m-m')$ with negligible probability $< 2^{\frac{2\ell}{3}}/2^\ell = 2^{\frac{-\ell}{3}}$. Second, we observe that the only other avenue for these values to be prime is if e is a multiple of $(m - m')$, which cannot be the case if e is prime. The probability that e is prime via Theorem 2.4 is at least $\frac{1}{\ell}$. □

B Completing the RSA Proof of Security

B.1 Proof of Lemma B.1

Lemma B.1. *The probability that there are less than q prime numbers in a set of $2\ell[q + \lambda]$ independent, randomly chosen ℓ-bit numbers is $\leq e^{-[q+\lambda](\frac{1}{2})^2}$.*

Proof. Let p be the probability that a random ℓ-bit number is a prime. Let X_i be a random variable representing whether an ℓ-bit number is prime, where $\Pr[X_i = 1] = p$ and $\Pr[X_i = 0] = 1 - p$. Let X be the sum of n X_i and $\mu = E[X] = \sum_{i=1}^n X_i$ (i.e., the expected number of primes in a set of n randomly chosen ℓ-bit numbers.)

To ensure that there are not too few primes, we bound this probability using Chernoff bounds (lower tail) where $0 < \delta \leq 1$

$$\Pr[X < (1-\delta)\mu] < e^{\frac{-\mu\delta^2}{2}}.$$

Set $\delta = \frac{1}{2}$ and $n = 2\ell[q + \lambda]$, which implies that $\mu = \sum_{i=1}^n X_i = 2\ell p[q + \lambda]$. Plugging this into the formula, we have

$$\Pr[X < \ell p[q + \lambda]] < e^{-\ell p[q+\lambda](\frac{1}{2})^2}.$$

By the Prime Number Theorem (Theorem 2.4), we know that $p \geq \frac{1}{\ell}$ and thus $\ell p \geq 1$. Note that by substituting 1 for ℓp, we get a looser bound of

$$\Pr[X < q + \lambda] < e^{-[q+\lambda](\frac{1}{2})^2}$$

which guarantees that less than $q + \lambda$ primes are seen after n trials with only negligible probability. □

B.2 Proof of Claim B.2

Claim B.2. *In the proof of Lemma 3.3, the challenge exponent e^* and the simulator produced value of $2^{\lambda - \lceil \lg(\tilde{i}) \rceil}(x - x^*) \prod_{j \in E}^{j \neq i^*} e_j$ are relatively prime with high probability.*

Proof. Recall that e^* is a random odd prime and thus is relatively prime to $2^{\lambda - \lceil \lg(\tilde{i}) \rceil}$. Based on the range of the chameleon hash function, we know that $(x - x^*) \in \{0,1\}^{\frac{2\ell}{3}}$. We know that e^* was chosen randomly from $\{0,1\}^\ell$. Thus, the chance that e^* falls into the small range of $\{0,1\}^{\frac{2\ell}{3}}$ (let alone collides with $(x - x^*)$) is negligible, specifically $2^{\frac{2\ell}{3}}/2^\ell = 2^{\frac{-\ell}{3}}$. Finally, we argue that for all j, we have $e^* \neq e_j$ with high probability. Recall that each e_j is prime and $e_j = c \oplus F_K(j)$, for some value $c \in \{0,1\}^\ell$ and random PRF key K. If $e^* = e_j$, then this implies that $F_K(i^*) = F_K(j)$

for $j \neq i^*$ and $i^*, j \leq t$. For a truly random function, this would happen with probability at most $t^2/2^\ell$, which is negligible in the security parameter. Thus, if this event occurs with non-neglibile probability using F then this admits a distinguishing attack against the PRF. *We emphasize that while we give out the PRF key K, this distinguishing attack does not require K in any way.* □