Short Unique Signatures from RSA with a Tight Security Reduction (in the Random Oracle Model)

Hovav Shacham

UC San Diego and UT Austin

Abstract. A signature scheme is unique if for every public key and message there is only one signature that is accepted as valid by the verification algorithm. At Crypto 2017, Guo, Chen, Susilo, Lai, Yang, and Mu gave a unique signature scheme whose security proof incurred a security loss logarithmic in the number of hash oracle queries made by the adversary, bypassing an argument due to Bader, Jager, Li, and Schäge that the security loss must be at least linear in the number of signing oracle queries made by the adversary. Unfortunately, the number of elements in a Guo et al. signature is also logarithmic in the number of hash oracle queries made by the adversary.

We translate Guo et al.'s signatures into the integer factorization setting. Doing so allows us to bring to bear signature aggregation ideas due to Lysyanskaya, Micali, Reyzin, and Shacham. We obtain unique signatures that are short *and* have a tight security reduction from the RSA problem.

1 Introduction

A signature scheme is unique if for every public key and message there is only one signature that is accepted as valid by the verification algorithm. Unique signatures make useful building blocks in primitives such as verifiable random functions [16].

At Eurocrypt 2016, Bader et al. [1] used a meta-reduction technique to show that any security proof for a unique signature scheme given some static assumption must incur a q_S security loss, where q_S is the number of signing oracle queries made by the adversary. In other words, ϵ-intractability of the underlying hard problem will translate only into $q_S\epsilon$-unforgeability of the unique signature. By contrast, there exist signature schemes with just *two* valid signatures per message whose security reductions incur a security loss of 2 [8].

At Crypto 2017, Guo et al. [9] observed that the Bader et al. meta-reduction made an assumption about how the unforgeability reduction would break the underlying hard problem: that it would extract the information it needed from the adversary's forgery. An unforgeability reduction that instead extracts the information it needs from the adversary's hash oracle queries (in the random oracle model) would fall outside the Bader et al. model and might be able to give better security guarantees.

Guo et al. presented a unique signature scheme with a proof of security assuming the computational Diffie-Hellman problem is intractable with a security loss of just $nq_H^{1/n}$, where q_H is the number of hash oracle queries made by the forger and n is a scheme parameter. Security loss is minimized by choosing $n \approx \ln q_H$. Unfortunately, signatures in Guo et al.'s scheme have length linear in n; Guo et al. describe the resulting signatures as "somewhat impractical even taking the loss factor into account," but "theoretically interesting."

Our contributions. We show that the ideas of Guo et al. give rise not just to "theoretically interesting" unique signatures with tight reductions but also to practical ones. We give a unique signature scheme with the same security loss as that of Guo et al. — $nq_H^{1/n}$ — where signatures consist of just two group elements, compared to $n + 1$ for Guo et al.

Guo et al. signatures consist of $n+1$ iterations of an underlying BLS signature. A BLS signature on a message M is of the form $H(M)^x$, where x is the secret exponent. It is straightforward to replace BLS with RSA full-domain hash signatures, of the form $H(M)^d \bmod N$, where d is a secret exponent.

Our key insight is that although BLS and RSA signatures have a similar signing operations but quite different verification operations. BLS verification transforms a signature in group G_1 into a value in group G_{T} by means of the pairing. It is intractable to go in the other direction, from G_{T} to G_1 [17]. RSA verification computes $\sigma^e \bmod N$, where e is the public exponent, producing a value that is still in $\mathbb{Z}/N\mathbb{Z}$ and therefore might be transformed into another signature to verify. This property was used by Lysyanskaya et al. [12] to build aggregate signatures from RSA. In an aggregate signature, a single, short signature takes the place of n signatures by n signers on n respective messages. A verifier given the aggregate signature σ, public keys $pk_1, \ldots, pk_n$, and messages $M_1, \ldots, M_n$ should accept only if the signing key corresponding to each pk_i was applied to M_i in the signing protocol.

The ideas of Lysyanskaya et al. do not immediately apply to give a short variant of Guo et al.'s signatures. The key difference is that in an aggregate signature scheme the verifier must still be sent the messages $M_1, \ldots, M_n$; in Guo et al.'s scheme, M_i includes $\sigma_1 \| \sigma_2 \| \cdots \| \sigma_{i-1}$; transmitting even just M_n along with the signature would negate any benefit to be had from signature aggregation.

We replace $\sigma_1 \| \sigma_2 \| \cdots \| \sigma_{i-1}$ in Guo et al.'s block messages with $G(M, 1, \sigma_1) \oplus G(M, 2, \sigma_2) \oplus \cdots \oplus G(M, i-1, \sigma_{i-1})$, essentially aggregating the block messages along with the block signatures. The Guo et al. security analysis no longer applies, and we replace it with a new security analysis that draws on both Guo et al. and Lysyanskaya et al.

The ith block signature in our scheme is of the form

$$\sigma_i = \left[\sigma_{i-1} + H(M, i, \mu_{i-1})\right]^d \bmod N$$

along with the block messages aggregated as as above. The verifier can compute μ_{i-1} as $\mu_i \oplus G(M, i, \sigma_i)$, then peel back σ_i to recover σ_{i-1} as $\sigma^e - H(M, i, \mu_{i-1}) \bmod N$. Verification consists of repeating this procedure n times starting from (σ_n, μ_n).

Our scheme resembles a construction for single-signer aggregate signatures with message recovery presented (without security proof) by Neven [14].

We show how our scheme can be instantiated from any family of certified trapdoor permutations, then explain how to obtain a suitable permutation from RSA. Our scheme shows that unique signatures built using the Guo et al. paradigm can be of more than just theoretical interest. At the 128-bit security level, and assuming $q_S = 2^{40}$ and $q_H = q_G = 2^{80}$, a signature in our scheme is under 4,000 bits, including 256 bits for the group E, whereas full-domain RSA signatures must be nearly 6,000 bits long to obtain $(128 + 40)$-bit factoring security and make up for their loose security reduction. (These estimates make use of the bit-length formula of Orman and Hoffman [15].)

2 Preliminaries

Trapdoor permutations. Let D be a finite set. A permutation family Π over D specifies a randomized algorithm for generating (descriptions of) a permutation and its inverse, written $(s, t) \xleftarrow{R} \mathit{Generate}$; an evaluation algorithm $\mathit{Evaluate}(s, \cdot)$; and an inversion algorithm $\mathit{Invert}(t, \cdot)$. We require that, for all (s, t) output by *Generate*, $\mathit{Evaluate}(s, \cdot)$ be a permutation of D, and that $\mathit{Invert}\big(t, \mathit{Evaluate}(s, \cdot)\big)$ be the identity map.

A trapdoor permutation family is one way if it is hard to invert given just the forward permutation description s. Formally [12, Definition 2.1], a trapdoor permutation family is (t, ϵ)-one way if no t-time algorithm $\mathcal{A}$ has advantage greater than ϵ in the game

$$\mathsf{Adv}\,\mathsf{Invert}_{\mathcal{A}} \stackrel{\text{def}}{=} \Pr\Big[x = \mathcal{A}\big(s, \mathit{Evaluate}(s, x)\big) : (s, t) \xleftarrow{R} \mathit{Generate}, x \xleftarrow{R} D\Big] \ ,$$

where the probability is over the coin tosses of *Generate* and $\mathcal{A}$.

A trapdoor permutation is certified [3] if a permutation description s output by *Generate* can be efficiently recognized: if there is an algorithm *Certify* that returns 1 for every string in the set $S = \big\{s \bigm| (s, t) \xleftarrow{R} \mathit{Generate}\big\}$ and 0 for every string not in the set S.

Where it does not introduce ambiguity, we prefer a more compact notation: we write $(\pi, \pi^{-1}) \xleftarrow{R} \Pi$ in place of $(s, t) \xleftarrow{R} \mathit{Generate}$ and $\pi(\cdot)$ and $\pi^{-1}(\cdot)$ in place of $\mathit{Evaluate}(s, \cdot)$ and $\mathit{Invert}(t, \cdot)$ respectively.

Digital signatures. A digital signature scheme consists of a randomized key generation algorithm that emits a (public) verification and a (private) signing key, written $(pk, sk) \xleftarrow{R} \mathit{KeyGen}$; a (possibly randomized) signing algorithm that takes a signing key and a message $M \in \{0,1\}^*$, and emits a signature σ, written $\sigma \xleftarrow{R} \mathit{Sign}(sk, M)$; and a (usually not randomized) verification algorithm that takes a verification key, message, and claimed signature, and returns 0 or 1, written $1 \stackrel{?}{=} \mathit{Verify}(pk, M, \sigma)$. We require that for all $(pk, sk) \xleftarrow{R} \mathit{KeyGen}$, for all $M \in \{0,1\}^*$, and for all $\sigma \xleftarrow{R} \mathit{Sign}(sk, M)$ we have $\mathit{Verify}(pk, M, \sigma) = 1$.

A digital signature scheme is (t, q_S, ϵ) existentially unforgeable if no t-time algorithm $\mathcal{A}$ has advantage greater than ϵ in the game

$$\mathsf{Adv}\,\mathsf{Forge}_{\mathcal{A}} \stackrel{\text{def}}{=} \Pr\left[\begin{array}{l} \mathit{Verify}(pk, M^*, \sigma^*) = 1 : \\ \qquad (pk, sk) \stackrel{\text{R}}{\leftarrow} \mathit{KeyGen},\ (M^*, \sigma^*) \stackrel{\text{R}}{\leftarrow} \mathcal{A}^{\mathit{Sign}(sk,\cdot)}(pk) \end{array}\right],$$

where the probability is over the coin tosses of *KeyGen*, *Sign*, and $\mathcal{A}$, and where we require that $\mathcal{A}$ make no more than q_S queries to the signing oracle and (to exclude trivial forgeries) that $\mathcal{A}$ not have queried its signing oracle at M^*.

In the random oracle model, all parties have access to a hash oracle that returns an independently random result on every input. A digital signature scheme is (t, q_H, q_S, ϵ) if no t-time algorithm $\mathcal{A}$ has advantage greater than ϵ in game $\mathsf{Adv}\,\mathsf{Forge}_{\mathcal{A}}$ above while making at most q_S signing oracle queries and at most q_H hash oracle queries. The definition generalizes naturally to multiple oracle hash functions.

A digital signature is unique if for every message $M \in \{0,1\}^*$ and every public key pk there is at most one signature σ such that $\mathit{Verify}(pk, M, \sigma) = 1$. This property must hold unconditionally, even for maliciously generated public keys that would not be emitted by *KeyGen*. In a unique signature scheme the signing and verification algorithms are not randomized.

Full-domain hash signatures. Trapdoor permutations give rise to a simple, natural unique signature scheme. Let Π be a trapdoor permutation family over domain D, and let $H\colon \{0,1\}^* \to D$ be a hash function, modeled as a random oracle. The key generation algorithm *KeyGen* picks a trapdoor permutation $(s, t) \stackrel{\text{R}}{\leftarrow} \mathit{Generate}$ and sets $pk = s$ and $sk = t$. The signing algorithm $\mathit{Sign}(sk, M)$ parses sk as t and emits $\sigma = \mathit{Invert}\big(t, H(M)\big)$. The verification algorithm $\mathit{Verify}(pk, M, \sigma)$ parses pk as s and rejects if $\mathit{Certify}(s) \neq 1$; checks that σ is an element of D and rejects if not; and, finally, accepts if $\mathit{Evaluate}(s, \sigma) = H(M)$ and rejects otherwise.

In our compact notation, the signature on a message M is $\sigma = \pi^{-1}\big(H(M)\big)$; the verifier checks whether $\pi(\sigma) \stackrel{?}{=} H(M)$.

Certifying that s is valid ensures that $\mathit{Evaluate}(s, \cdot)$ is a permutation of D; there can be only one element of D whose image under that permutation is $H(M)$, which guarantees unique signatures.

Bellare and Rogaway showed that the full-domain hash signature scheme is existentially unforgeable in the random oracle model if the underlying trapdoor permutation is one-way [2]. Their reduction suffered a security loss of q_H. Coron gave an improved security analysis, with security loss q_S, assuming that the underlying trapdoor permutation family is homomorphic, as RSA is [6]. Coron later gave a "meta-reduction" that showed that any proof that full-domain hash signatures are secure assuming that an underlying trapdoor permutation family is one way must incur a q_S security loss [7]. Even so, Kakvi and Kiltz proved that RSA full-domain hash signatures are secure under the phi-hiding assumption with a *tight* reduction [10]. Kakvi-Kiltz signatures have public exponent $e < N^{1/4}$. With e in this range, the RSA permutation is not certified, and the resulting signatures are not unique.

3 Unique Signatures with Tight Security Reduction

At Eurocrypt 2016, Bader et al. [1] extended Coron's meta-reduction technique to show that any security proof for a unique signature scheme assuming a static assumption like computational Diffie-Hellman or the trapdoor permutation one-wayness must incur a q_S security loss. Nevertheless, at Crypto 2017, Guo et al. [9] were able to present unique signatures secure under the computational Diffie-Hellman problem, with a tight security reduction. The Bader et al. metareduction assumes that the simulator extracts the information it uses to break the underlying problem from the adversary's forgery. In the Guo et al. signature scheme, the simulator instead extracts that information from the adversary's *hash queries*. A Guo et al. signature is built from $n+1$ blocks, where n is a system parameter. Each block consists of a BLS signature [4] on the message and the previous blocks. For the adversary to compute the ith block of a forgery in progress, it must first reveal blocks 1 through $i-1$ in a hash query; the simulator takes advantage of these hash queries to solve the underlying hard problem.

The security loss in the Guo et al. reduction is $nq_H^{1/n}$. Even $n = 2$ improves on the Bader et al. bound; with $q_H = 2^{80}$, setting $n = 55$ gives security loss less than 151.[1]

Guo et al. observe that their signature framework could be instantiated using a different underlying block signature. In this section, we translate the Guo et al. signatures to the trapdoor permutation setting, still with signature size linear in n. In the next section, we present our variant of Guo et al. signatures with $O(1)$ signature size.

Let Π be a trapdoor permutation family over domain D. Let $H\colon \{0,1\}^* \times \mathbb{N} \times D^* \to D$ be a hash function, modeled as a random oracle. Note that we can instantiate H using a hash function $H'\colon \{0,1\}^* \to D$ by means of an unambiguous encoding of $\{0,1\}^* \times \mathbb{N} \times D^*$ in $\{0,1\}^*$.

KeyGen. Pick $(s,t) \xleftarrow{\mathrm{R}} \mathit{Generate}$. The public key is $pk = s$. The private key is $sk = t$.

Sign(sk, M)**.** Parse sk as t. For each i, $1 \le i \le n+1$, compute

$$h_i \leftarrow H\big(M, i, (\sigma_1, \sigma_2, \ldots, \sigma_{i-1})\big) \qquad \text{and} \qquad \sigma_i \leftarrow \mathit{Invert}(t, h_i) \ .$$

The signature is $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n, \sigma_{n+1})$.

Verify(pk, M, σ)**.** Parse pk as s and reject if $\mathit{Certify}(s) \neq 1$. Parse σ as $(\sigma_1, \sigma_2, \ldots, \sigma_n, \sigma_{n+1}) \in D^{n+1}$, and reject if parsing fails. For each i, $1 \le i \le n+1$, compute

$$h_i \leftarrow H\big(M, i, (\sigma_1, \sigma_2, \ldots, \sigma_{i-1})\big) \ .$$

For each i, $i \le i \le n+1$, check that

$$\mathit{Evaluate}(s, \sigma_i) = h_i \ .$$

If any of these checks fails, reject; otherwise, accept.

[1] The Bitcoin network hash rate is estimated at 2^{79} hashes per day. See `https://blockchain.info/charts/hash-rate`, visited September 22, 2017.

The proof that the Guo et al. signature scheme is unforgeable [9, Section 5] can be easily adapted to show that the variant above is unforgeable as well. The simulator $\mathcal{B}$ is given the public description π of a trapdoor permutation, and an element y^* of D. Its goal is to find $x^* \in D$ such that $\pi(x^*) = y^*$. It sets π as the challenge signing key and interacts with the forger $\mathcal{A}$ as specified by Guo et al. For hash oracle queries other than the one in which it will embed the challenge, the simulator chooses $x \xleftarrow{R} D$ and returns $\pi(x)$ as the hash; this allows it to compute the corresponding block signature x, should it later need to answer a signing oracle query on the same message. For the hash oracle query where it chooses to embed the challenge, $\mathcal{A}$ simply responds with y^*. A subsequent hash oracle query from $\mathcal{B}$ that contains the next block signature on the same message will reveal x^* to $\mathcal{B}$. The proof and analysis are otherwise unchanged.

4 Short Unique Signatures with Tight Security Reduction

We now explain how to compress the signatures of Guo et al. Our scheme is inspired by the sequential aggregate signature of scheme of Lysyanskaya et al. [12].

Let D be a group with operation $+$, identity 0_D, and inverse operation $-$. Let Π be a trapdoor permutation family over domain D; we do not require any homomorphic interaction between Π and $+$.

Let λ be a positive integer (whose value depends on the security parameter, as discussed below), and let E be the set $\{0,1\}^\lambda$ together with the bitwise exclusive or operation $\oplus$ and identity 0_E.

Let $H\colon \{0,1\}^* \times \mathbb{N} \times E \to D$ and $G\colon \{0,1\}^* \times N \times D \to E$ be hash functions, modeled as random oracles. As before, we can instantiate H and G from hash functions with domain $\{0,1\}^*$ using appropriate unambiguous encodings.

KeyGen. Pick $(s,t) \xleftarrow{R} \mathit{Generate}$. The public key is $pk = s$. The private key is $sk = t$.

Sign(sk, M)**.** Parse sk as t. Set

$$\sigma_0 \leftarrow 0_D \qquad \text{and} \qquad \mu_0 \leftarrow 0_E \ .$$

For each i, $1 \le i \le n$, compute

$$\sigma_i \leftarrow \mathit{Invert}\big(t, \sigma_{i-1} + H(M, i, \mu_{i-1})\big) \tag{1}$$

and

$$\mu_i \leftarrow \mu_{i-1} \oplus G(M, i, \sigma_i) \ . \tag{2}$$

The signature is $\sigma = (\sigma_n, \mu_n)$.

Verify(pk, M, σ)**.** Parse pk as s and reject if $\mathit{Certify}(s) \neq 1$. Parse σ as $(\sigma_n, \mu_n) \in D \times E$, and reject if parsing fails. For i from n down to 1, compute

$$\mu_{i-1} \leftarrow \mu_i \oplus G(M, i, \sigma_i) \tag{3}$$

and

$$\sigma_{i-1} \leftarrow \mathit{Evaluate}(s, \sigma_i) - H(M, i, \mu_{i-1}) \ . \tag{4}$$

Check that

$$\sigma_0 = 0_D \qquad \text{and} \qquad \mu_0 = 0_E \ .$$

If either of these checks fails, reject; otherwise, accept.

4.1 Proof of Correctness

Let $\sigma = (\sigma_n, \mu_n)$ be a signature on message M under keys $pk = s$ and $sk = t$. The signer computed partial values $\sigma_0, \sigma_1, \ldots, \sigma_n$ and $\mu_0, \mu_1, \ldots, \mu_n$ according to (1) and (2). The verifier will compute partial values $\sigma'_n, \sigma'_{n-1}, \ldots, \sigma'_0$ and $\mu'_n, \mu'_{n-1}, \ldots, \mu'_0$ according to (3) and (4).

We know that $\sigma'_n = \sigma_n$ and $\mu'_n = \mu_n$, because the verifier is verifying the output of the signing algorithm. Suppose that for some i we have $\sigma'_i = \sigma_i$ and $\mu'_i = \mu_i$. Then

$$\mu'_{i-1} = \mu'_i \oplus G(M, i, \sigma'_i) = \mu_i \oplus G(M, i, \sigma_i) = \mu_{i-1}$$

and

$$\begin{aligned}
\sigma'_{i-1} &= \mathit{Evaluate}(s, \sigma'_i) - H(M, i, \mu'_{i-1}) = \mathit{Evaluate}(s, \sigma_i) - H(M, i, \mu_{i-1}) \\
&= \mathit{Evaluate}\Big(s, \mathit{Invert}\big(t, \sigma_{i-1} + H(M, i, \mu_{i-1})\big)\Big) - H(M, i, \mu_{i-1}) \\
&= \sigma_{i-1} + H(M, i, \mu_{i-1}) - H(M, i, \mu_{i-1}) = \sigma_{i-1} \ .
\end{aligned}$$

By induction, then, $\sigma'_0 = \sigma_0 = 0_D$ and $\mu'_0 = \mu_0 = 0_E$, so the verification checks will succeed.

4.2 Proof of Uniqueness

Suppose for the sake of contradiction that $\sigma \neq \sigma'$ are two valid signatures on a message M under public key pk. If pk cannot be parsed as s or $\mathit{Certify}(s) \neq 1$, the verification algorithm will reject both σ and σ'. Accordingly, $\mathit{Evaluate}(s, \cdot)$ must be a permutation of D, which means that, for any $u, v \in D$, if $\mathit{Evaluate}(s, u) = \mathit{Evaluate}(s, v)$ then $u = v$.

Parse σ as $(\sigma_n, \mu_n) \in D \times E$ and σ' as $(\sigma'_n, \mu'_n) \in D \times E$. If either signature fails to parse it will be rejected by the verification algorithm. Applied to σ, the verification algorithm will compute partial values $\sigma_n, \sigma_{n-1}, \ldots, \sigma_0$ and $\mu_n, \mu_{n-1}, \ldots, \mu_0$ according to (3) and (4). Applied to σ', the verification algorithm will compute partial values $\sigma'_n, \sigma'_{n-1}, \ldots, \sigma'_0$ and $\mu'_n, \mu'_{n-1}, \ldots, \mu'_0$ according to (3) and (4). All of the σ_i and σ'_i values must be elements of D because σ_n and σ'_n are. All of the μ_i and μ'_i values must be elements of E because μ_n and μ'_n are.

Since both σ and σ' verify, we know that $\sigma_0 = 0_D$, $\mu_0 = 0_E$, $\sigma'_0 = 0_D$, and $\mu'_0 = 0_E$. Stated another way, we know that $\sigma_0 = \sigma'_0$ and $\mu_0 = \mu'_0$. Now suppose that for some i we have $\sigma_{i-1} = \sigma'_{i-1}$ and $\mu_{i-1} = \mu'_{i-1}$. Then, substituting (4) twice in $\sigma_{i-1} = \sigma'_{i-1}$ we obtain

$$\mathit{Evaluate}(s, \sigma_i) - H(M, i, \mu_{i-1}) = \mathit{Evaluate}(s, \sigma'_i) - H(M, i, \mu'_{i-1}) \ ,$$

and, substituting $\mu'_{i-1} = \mu_{i-1}$ on the right hand side, we obtain

$$\mathit{Evaluate}(s, \sigma_i) - H(M, i, \mu_{i-1}) = \mathit{Evaluate}(s, \sigma'_i) - H(M, i, \mu_{i-1}) \ ,$$

and we can obtain

$$\mathit{Evaluate}(s, \sigma_i) = \mathit{Evaluate}(s, \sigma'_i)$$

by adding $H(M, i, \mu_{i-1})$ to both sides. Because $\mathit{Evaluate}(s, \cdot)$ is a permutation, we conclude that $\sigma_i = \sigma'_i$. Now, substituting (3) twice into $\mu_{i-1} = \mu'_{i-1}$, we have

$$\mu_i \oplus G(M, i, \sigma_i) = \mu'_i \oplus G(M, i, \sigma'_i)$$

and, since $\sigma_i = \sigma'_i$, we conclude that $G(M, i, \sigma_i) = G(M, i, \sigma'_i)$ and therefore $\mu_i = \mu'_i$. By induction, $\sigma_n = \sigma'_n$ and $\mu_n = \mu'_n$, contradicting the assumption that $\sigma \neq \sigma'$.

4.3 Proof of Unforgeability

Suppose, for the sake of contradiction, that there exists some algorithm $\mathcal{A}$ that forges signatures with non-negligible probability. We will show that we can use $\mathcal{A}$ to break the one-wayness of the underlying trapdoor permutation family Π.

Description of the simulator. We describe algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the security of the trapdoor permutation family Π.

Environment setup. Algorithm $\mathcal{B}$ is given a permutation key s^* and a value $y^* \in D$. Its goal is to compute $x^* \in D$ such that $\mathit{Evaluate}(s^*, x^*) = y^*$. Algorithm $\mathcal{B}$ picks an integer c^* uniformly at random from the range $[1, n]$, and then an integer k^* uniformly at random from the range $\left[1, (q_H + 1)^{(n+1-c^*)/n}\right]$. (Note that the range from which k^* is chosen depends on c^*.) Algorithm $\mathcal{B}$ initializes a global variable x^* to $\perp$ and a global counter k to 0.

Algorithm $\mathcal{B}$ sets $pk \leftarrow s^*$; it does not know the corresponding sk. It then runs $\mathcal{A}$ with input pk.

The tables recording hash oracle queries and responses. Algorithm $\mathcal{B}$ maintains a table to help it answer H oracle queries, which we call the H-table. The H-table starts out empty. Each row in the H-table has the following entries:

- $M \in \{0,1\}^*$, $i \in \mathbb{N}$, $\mu \in E$: these, together, are the inputs to the hash query.
- $z \in D$: the output from the hash query.
- *good* $\in \{\texttt{true}, \texttt{false}\}$: a flag used by $\mathcal{B}$ to track hash queries that could contribute to an eventual signature.
- *internal* $\in \{\texttt{true}, \texttt{false}\}$: a flag used by $\mathcal{B}$ to track H oracle queries that it initiated as part of signature generation, versus those initiated by $\mathcal{A}$.
- $x \in D \cup \{\perp\}$: a secret used by $\mathcal{B}$ as the basis for computing the answer z to some hash queries.

Algorithm $\mathcal{B}$ likewise maintains a table to help it answer G oracle queries, which we call the G-table. The G-table starts out empty. Each row in the G-table has the following entries:

- $M \in \{0,1\}^*$, $i \in \mathbb{N}$, $\sigma \in D$: these, together, are the inputs to the hash query.
- $z \in E$: the output from the hash query.
- *good* $\in \{\texttt{true}, \texttt{false}\}$: a flag used by $\mathcal{B}$ to track hash queries that could contribute to an eventual signature.
- *internal* $\in \{\texttt{true}, \texttt{false}\}$: a flag used by $\mathcal{B}$ to track G oracle queries that it initiated as part of signature generation, versus those initiated by $\mathcal{A}$.
- $\mu \in E \cup \{\perp\}$: the value algorithm $\mathcal{A}$ is expected to compute for μ_i, as the xor of μ_{i-1} and z, or $\perp$ if not known.

Answering an H oracle query. To answer an H oracle query on $(M, i, \mu) \in \{0,1\}^* \times \mathbb{N} \times E$, $\mathcal{B}$ responds as follows.

1. If there has already been an H oracle query for (M, i, μ), there will be an entry $(M, i, \mu, z, \mathit{good}, \mathit{internal}, x)$ in the H-table. Algorithm $\mathcal{B}$ responds with z. This keeps the oracle consistent if queried multiple times on the same input.
2. If $i < 1$ or $i > n$, the query is not relevant to any signature. Algorithm $\mathcal{B}$ picks $z \xleftarrow{\text{R}} D$ at random, sets $\mathit{good} \leftarrow \texttt{false}$, $\mathit{internal} \leftarrow \texttt{false}$, and $x \leftarrow \perp$. It adds the entry $(M, i, \mu, z, \mathit{good}, \mathit{internal}, x)$ to the H-table, and responds with z.
3. If $i = 1$, $\mathcal{B}$ consults μ. If $\mu \neq 0_E$, the query is not relevant to any signature. Algorithm $\mathcal{B}$ picks $z \xleftarrow{\text{R}} D$ at random, sets $\mathit{good} \leftarrow \texttt{false}$, $\mathit{internal} \leftarrow \texttt{false}$, and $x = \perp$. It adds the entry $(M, i, \mu, z, \mathit{good}, \mathit{internal}, x)$ to the H-table, and responds with z.
   Otherwise, $\mu = 0$, and the query is relevant to an eventual signature on the message M. Algorithm $\mathcal{B}$ will decide whether to embed its challenge as the answer to this query, according to the following criteria. If this H oracle query was generated internally by $\mathcal{B}$ as part of handling a signing query from $\mathcal{A}$, algorithm $\mathcal{B}$ sets $\mathit{internal} \leftarrow \texttt{true}$; it will not embed its challenge as the answer to this query. Otherwise, $\mathcal{B}$ sets $\mathit{internal} \leftarrow \texttt{false}$. If $i \neq c^*$, $\mathcal{B}$ will not embed its challenge as the answer to this query. Otherwise, if $i = c^*$, $\mathcal{B}$ increments the global counter k. If the counter k now has a value different from k^*, $\mathcal{B}$ will not embed its challenge as the answer to this query. Otherwise all three of the following conditions hold: (1) the query was generated by $\mathcal{A}$; (2) i equals c^*; and (3) k, incremented, equals k^*. In this case $\mathcal{B}$ *will* embed its challenge as the answer to this query.
   If $\mathcal{B}$ didn't chose to embed its challenge as the answer to this query, it selects $x \xleftarrow{\text{R}} D$, computes $z \leftarrow \mathit{Evaluate}(s^*, x)$, and sets $\mathit{good} \leftarrow \texttt{true}$. It adds the entry $(M, i, \mu, z, \mathit{good}, \mathit{internal}, x)$ to the H-table, and responds with z.
   If $\mathcal{B}$ did chose to embed its challenge as the answer to this query, it sets $x \leftarrow \perp$, $z \leftarrow y^*$, and $\mathit{good} \leftarrow \texttt{true}$. It adds the entry $(M, i, \mu, z, \mathit{good}, \mathit{internal}, x)$ to the H table.

Before returning, algorithm $\mathcal{B}$ checks whether its answer to this query is inconsistent with a previous query to the G oracle at level i. Algorithm $\mathcal{B}$ examines all entries in the G-table matching $(M'' = M,\ i'' = i,\ \sigma'',\ z'',\ \mathit{good}'' = \texttt{false},\ \mathit{internal}'',\ \mu'')$. If $x \neq \perp$, algorithm $\mathcal{B}$ checks for such an entry with $\sigma'' = x$. If $x = \perp$, algorithm $\mathcal{B}$ checks for such an entry with $\mathit{Evaluate}(s^*, \sigma'') = y^*$. In either case there is at most one such entry in the G-table (in the latter case because $\mathit{Evaluate}(s^*, \cdot)$ is a permutation). If such an entry exists, it was inserted with $\mathit{good} = \texttt{false}$ but should have been inserted with $\mathit{good} = \texttt{true}$. Algorithm $\mathcal{B}$ cannot fix this problem and must abort.

If algorithm $\mathcal{B}$ was not forced to abort, it responds to the H-query with z.

4. Otherwise, we have $2 \leq i \leq n$. Algorithm $\mathcal{B}$ cannot immediately tell whether μ makes the query good — it must consult G oracle queries for the previous block, $i-1$. Algorithm $\mathcal{B}$ searches the G-table for an entry matching $(M' = M,\ i' = i-1,\ \sigma',\ z',\ \mathit{good}' = \texttt{true},\ \mathit{internal}',\ \mu' = \mu)$. There is at most one such entry in the G-table.

   If no matching entry is found, the query is not relevant to any signature. Algorithm $\mathcal{B}$ picks $z \stackrel{\mathrm{R}}{\leftarrow} D$ at random, sets $\mathit{good} \leftarrow \texttt{false}$, $\mathit{internal} \leftarrow \texttt{false}$, and $x = \perp$. It adds the entry $(M, i, \mu, z, \mathit{good}, \mathit{internal}, x)$ to the H-table, and responds with z.

   Otherwise, there is a matching entry $(M' = M,\ i' = i-1,\ \sigma',\ z',\ \mathit{good}' = \texttt{true},\ \mu' = \mu)$. (For each M' and i' there can be at most one entry in the G-table with $\mathit{good}' = \texttt{true}$.) This means that the H query now being handled *is* relevant to an eventual signature on the message M. Algorithm $\mathcal{B}$ will decide whether to embed its challenge as the answer to this query according to the following criteria. If this H oracle query was generated internally by $\mathcal{B}$ as part of handling a signing query from $\mathcal{A}$, algorithm $\mathcal{B}$ sets $\mathit{internal} \leftarrow \texttt{true}$; it will not embed its challenge as the answer to this query. Otherwise, $\mathcal{B}$ sets $\mathit{internal} \leftarrow \texttt{false}$. If $i \neq c^*$, $\mathcal{B}$ will not embed its challenge as the answer to this query. Otherwise, if $i = c^*$, $\mathcal{B}$ increments the global counter k. If the counter k now has a value different from k^*, $\mathcal{B}$ will not embed its challenge as the answer to this query. Otherwise all three of the following conditions hold: (1) the query was generated by $\mathcal{A}$; (2) i equals c^*; and (3) k, incremented, equals k^*. In this case $\mathcal{B}$ *will* embed its challenge as the answer to this query.

   If $\mathcal{B}$ didn't chose to embed its challenge as the answer to this query, it selects $x \stackrel{\mathrm{R}}{\leftarrow} D$, computes $z \leftarrow \mathit{Evaluate}(s^*, x) - \sigma'$, and sets $\mathit{good} \leftarrow \texttt{true}$. It adds the entry $(M, i, \mu, z, \mathit{good}, \mathit{internal}, x)$ to the H table, and responds with z.

   If $\mathcal{B}$ did chose to embed its challenge as the answer to this query, it sets $x \leftarrow \perp$, $z \leftarrow y^* - \sigma'$, and $\mathit{good} \leftarrow \texttt{true}$. It adds the entry $(M, i, \mu, z, \mathit{good}, \mathit{internal}, x)$ to the H table.

   Before returning, algorithm $\mathcal{B}$ checks whether its answer to this query is inconsistent with a previous query to the G oracle at level i. Algorithm $\mathcal{B}$ examines all entries in the G-table matching $(M'' = M,\ i'' = i,\ \sigma'',\ z'',\ \mathit{good}'' = \texttt{false},\ \mathit{internal}'',\ \mu'')$. If $x \neq \perp$, algorithm $\mathcal{B}$ checks for such an entry with $\sigma'' = x$. If $x = \perp$, algorithm $\mathcal{B}$ checks for such an entry with $\mathit{Evaluate}(s^*, \sigma'') = y^*$. In either case there is at most one such entry in the

G-table (in the latter case because $\mathit{Evaluate}(s^*, \cdot)$ is a permutation). If such an entry exists, it was inserted with $\mathit{good} = \texttt{false}$ but should have been inserted with $\mathit{good} = \texttt{true}$. Algorithm $\mathcal{B}$ cannot fix this problem and must abort.
If algorithm $\mathcal{B}$ was not forced to abort, it responds to the H-query with z.

Answering a G oracle query. To answer a G oracle query on $(M, i, \sigma) \in \{0,1\}^* \times \mathbb{N} \times D$, $\mathcal{B}$ responds as follows.

1. If there has already been a G oracle query for (M, i, σ), there will be an entry $(M, i, \sigma, z, \mathit{good}, \mu)$ in the G-table. Algorithm $\mathcal{B}$ responds with z. This keeps the oracle consistent if queried multiple times on the same input.
2. If $i < 1$ or $i > n$, the query is not relevant to any signature. Algorithm $\mathcal{B}$ picks $z \xleftarrow{R} D$ at random. It sets $\mathit{good} \leftarrow \texttt{false}$, $\mathit{internal} \leftarrow \texttt{false}$, and $\mu \leftarrow \perp$. It adds the entry $(M, i, \sigma, z, \mathit{good}, \mathit{internal}, \mu)$ to the G-table, and responds with z.
3. Otherwise, we have $i \leq i \leq n$. Algorithm $\mathcal{B}$ searches its H-table for entries matching $(M' = M,\ i' = i,\ \mu',\ z',\ \mathit{good}' = \texttt{true},\ \mathit{internal}',\ x')$. There is at most one such entry in the H-table.
   If no matching entry is found, the query is not relevant to any signature. Algorithm $\mathcal{B}$ picks $z \xleftarrow{R} E$ at random. It sets $\mathit{good} \leftarrow \texttt{false}$, $\mathit{internal} \leftarrow \texttt{false}$, and $\mu \leftarrow \perp$. It adds the entry $(M, i, \sigma, z, \mathit{good}, \mathit{internal}, \mu)$ to the G-table, and responds with z.
   Otherwise, algorithm $\mathcal{B}$ has found a matching entry $(M' = M,\ i' = i,\ \mu',\ z',\ \mathit{good}' = \texttt{true},\ \mathit{internal}',\ x')$ in the H-table. The query now being handled is relevant to an eventual signature if either (a) $x' \neq \perp$ and $\sigma = x'$ or (b) $x' = \perp$ and $\mathit{Evaluate}(s^*, \sigma) = y^*$. In the latter case, algorithm $\mathcal{B}$ has just learned the solution to the inversion problem it was posed. It stores the solution σ in its global variable x^*.
   If the query now being handled is not relevant to any signature. Algorithm $\mathcal{B}$ picks $z \xleftarrow{R} E$ at random. It sets $\mathit{good} \leftarrow \texttt{false}$, $\mathit{internal} \leftarrow \texttt{false}$, and $\mu \leftarrow \perp$. It adds the entry $(M, i, \sigma, z, \mathit{good}, \mathit{internal}, \mu)$ to the G-table, and responds with z.
   Otherwise, the query now being handled *is* relevant to an eventual signature on the message M. Algorithm $\mathcal{B}$ picks $z \xleftarrow{R} E$ at random and sets $\mathit{good} \leftarrow \texttt{true}$ and $\mu \leftarrow \mu' \oplus z$. If this G oracle query was generated internally by $\mathcal{B}$ as part of handling a signing query from $\mathcal{A}$, algorithm $\mathcal{B}$ sets $\mathit{internal} \leftarrow \texttt{true}$. Otherwise, $\mathcal{B}$ sets $\mathit{internal} \leftarrow \texttt{false}$. It adds the entry $(M, i, \sigma, z, \mathit{good}, \mu)$ to the G-table.
   So long as $i \neq n$, algorithm $\mathcal{B}$ checks whether its answer to this query is inconsistent with a previous query at level $i+1$ before returning. Algorithm $\mathcal{B}$ searches its H-table for an entry matching $(M'' = M,\ i'' = i+1,\ \mu'' = \mu' \oplus z,\ z'',\ \mathit{good}'' = \texttt{false},\ \mathit{internal}'' = \texttt{false},\ x'')$. If such an entry exists, it was inserted with $\mathit{good} = \texttt{false}$ but should have been inserted with $\mathit{good} = \texttt{true}$. Algorithm $\mathcal{B}$ cannot fix this problem and must abort.
   If algorithm $\mathcal{B}$ was not forced to abort, it responds to the hash query with z.

Answering a signing oracle query. To answer a signature query on $M \in \{0,1\}^*$, $\mathcal{B}$ responds as follows.

Otherwise, for each i from 1 to n, algorithm $\mathcal{B}$ searches its H-table for entries of the form $(M' = M,\ i' = i,\ \mu',\ z',\ \mathit{good}' = \texttt{true},\ \mathit{internal}' = \texttt{false},\ x')$. There will be at most one such entry in the H-table for each i. There will be some index I such that for all $i \leq I$ there is a matching entry in the table, and for all $i > I$ there is not. (It's possible that there are no matching entries in the H-table, in which case I is 0.)

For each i from 1 to n, algorithm $\mathcal{B}$ searches its G-table for entries of the form $(M'' = M,\ i'' = i,\ \sigma'',\ z'',\ \mathit{good}'' = \texttt{true},\ \mathit{internal}'' = \texttt{false},\ \mu'')$. There will be at most one such entry in the G-table for each i. There will be some index J such that for all $i \leq J$ there is a matching entry in the table, and for all $i > J$ there is not. (It's possible that there are no matching entries in the G-table, in which case J is 0.) It must be the case that either $J = I - 1$ or $J = I$.

If $I = 0$, algorithm $\mathcal{B}$ sets $\sigma_I \leftarrow 0_D$ and $\mu_I \leftarrow 0_E$.

Otherwise, if $I > 0$ and $J = I$, let $(M' = M,\ i' = I,\ \mu',\ z',\ \mathit{good}' = \texttt{true},\ \mathit{internal}' = \texttt{false},\ x')$ be the matching entry in the G-table for $i' = I$, and let $(M'' = M,\ i'' = J,\ \sigma'',\ z'',\ \mathit{good}'' = \texttt{true},\ \mathit{internal}'' = \texttt{false},\ \mu'')$. be the matching entry in the G-table for $i'' = J = I$. Algorithm $\mathcal{B}$ sets $\sigma_I \leftarrow \sigma''$ and $\mu_I \leftarrow \mu''$.

Otherwise, we have $I > 0$ and $J = I - 1$. Let $(M' = M,\ i' = I,\ \mu',\ z',\ \mathit{good}' = \texttt{true},\ \mathit{internal}' = \texttt{false},\ x')$ be the matching entry in the G-table for $i' = I$. If $x' = \perp$, algorithm $\mathcal{B}$ does not know how to compute the next block signature, and must abort. Otherwise, $x' \in D$, and $\mathcal{B}$ sets $\sigma_I \leftarrow x'$. It makes an internal G oracle query for $G(M, i, \sigma_I)$. This query ensures that there is an entry in the G-table of the form $(M'' = M,\ i'' = I,\ \sigma'',\ z'',\ \mathit{good}'' = \texttt{true},\ \mathit{internal}'' = \texttt{true},\ \mu'')$. Algorithm $\mathcal{B}$ sets $\mu_I \leftarrow \mu''$.

Now algorithm $\mathcal{B}$ repeats the following steps for each i from $I + 1$ to n. It makes an internal H oracle query for $H(M, i, \mu_{i-1})$. This hash query ensures that there is an entry in the H-table of the form $(M' = M,\ i' = i,\ \mu' = \mu_{i-1},\ z',\ \mathit{good}' = \texttt{true},\ \mathit{internal}' = \texttt{true},\ x')$. Algorithm $\mathcal{B}$ sets $\sigma_i \leftarrow x'$. (Algorithm $\mathcal{B}$ has set μ_{i-1} to the value that will cause its hash oracle code to create an entry with $\mathit{good}' = \texttt{true}$ and $x' \neq \perp$.) Algorithm $\mathcal{B}$ makes an internal G oracle query for $G(M, i, \sigma_i)$. This query ensures that there is an entry in the G-table of the form $(M'' = M,\ i'' = i,\ \sigma'',\ z'',\ \mathit{good}'' = \texttt{true},\ \mathit{internal}'' = \texttt{true},\ \mu'')$. Algorithm $\mathcal{B}$ sets $\mu_i \leftarrow \mu''$.

When the loop has finished, $\mathcal{B}$ returns (σ_n, μ_n) as the answer to the signature query.

Handling the claimed forgery. Finally, algorithm $\mathcal{A}$ halts and emits a message $M^* \in \{0,1\}^*$ and a claimed signature forgery σ^* on M. Algorithm $\mathcal{A}$ must not have made a signing oracle query on message M^*, or the forgery would be trivial. Algorithm $\mathcal{B}$ attempts to parse σ as $(\sigma_n^*, \mu_n^*) \in D \times E$. If parsing succeeds, $\mathcal{B}$ attempts to verify the signature.

Algorithm $\mathcal{B}$ repeats the following steps for each i from n down to 1. Algorithm $\mathcal{B}$ checks that the G-table includes an entry of the form $(M'' = M^*,\ i'' =$

$i,\ \sigma'' = \sigma_i^*,\ z'',\ good'' = \texttt{true},\ internal'' = \texttt{false},\ \mu'')$. If there is no such entry, $\mathcal{A}$ must not have queried its G oracle at $G(M^*, i, \sigma_i^*)$. Algorithm $\mathcal{B}$ declares failure and aborts. Otherwise, $G(M^*, i, \sigma_i^*) = z''$. Algorithm $\mathcal{B}$ sets $\mu_{i-1}^* \leftarrow \mu_i^* \oplus z''$. Algorithm $\mathcal{B}$ then checks that the H-table includes an entry of the form $(M' = M^*,\ i' = i,\ \mu' = \mu_{i-1}^*,\ z',\ good' = \texttt{true},\ internal' = \texttt{false},\ x')$. If there is no such entry, $\mathcal{A}$ must not have queried its H oracle at $H(M^*, i, \mu_{i-1}^*)$. Algorithm $\mathcal{B}$ declares failure and aborts. Otherwise, $H(M^*, i, \mu_{i-1}^*) = z'$. Algorithm $\mathcal{B}$ sets $\sigma_{i-1}^* \leftarrow \mathit{Evaluate}(s, \sigma_i^*) - z'$ and continues to the next loop iteration.

Assuming it completes the loop without aborting, $\mathcal{B}$ can tell whether the claimed forgery σ^* is valid by checking whether $\sigma_0^* = 0_D$ and $\mu_0^* = 0_E$.

Finally, $\mathcal{B}$ checks whether one of $\mathcal{A}$'s hash queries revealed the answer to the one-wayness challenge posed to $\mathcal{B}$ by examining its global variable x^*. If that variable still contains $\perp$, $\mathcal{B}$ declares failure. Otherwise $\mathcal{B}$ declares success: x^* is the value such that $\mathit{Evaluate}(s^*, x^*) = y^*$.

Analysis of the simulator. We now analyze the performance of the simulator $\mathcal{B}$. Suppose that $\mathcal{A}$ makes q_S signing queries and q_H hash queries, and produces a valid forgery with probability ϵ. There are six reasons why $\mathcal{B}$ might fail to break the one-wayness of the trapdoor permutation family Π:

1. Algorithm $\mathcal{B}$ discovers, when handling a H oracle query at level i, that it failed to mark a G oracle query at level i as good.
2. Algorithm $\mathcal{B}$ discovers, when handling a G oracle query at level i, that it failed to mark a H oracle query at level $i+1$ as good.
3. Algorithm $\mathcal{B}$ discovers, when verifying $\mathcal{A}$'s claimed forgery, that $\mathcal{A}$ didn't query the H oracle at $H(M^*, i, \mu_{i-1}^*)$ for some i.
4. Algorithm $\mathcal{B}$ discovers, when verifying $\mathcal{A}$'s claimed forgery, that $\mathcal{A}$ didn't query the G oracle at $G(M^*, i, \sigma_i^*)$ for some i.
5. Algorithm $\mathcal{A}$ makes a signature query that $\mathcal{B}$ cannot answer because it would require knowing the preimage of the challenge value y^*.
6. Algorithm $\mathcal{A}$ produces a valid forgery but never made a hash oracle query that revealed the preimage of the challenge value y^*.

We can bound the likelihood of reasons 1 through 4 in a straightforward way. To bound the likelihood of reasons 5 and 6, we will need to recall machinery by Guo et al. [9].

We start by bounding reason 1. Algorithm $\mathcal{B}$ is handling an H oracle query for $H(M, i, \mu)$. For each M and i, there is exactly one value of μ that will trigger a consistency check, with $\mathcal{B}$ examining its G-table for entries matching $(M'' = M,\ i'' = i,\ \sigma'',\ z'',\ good'' = \texttt{false},\ internal'',\ \mu'')$. There is exactly one value of σ'' in such an entry that would cause $\mathcal{B}$ to abort, and that this value depends on a value that isn't in $\mathcal{A}$'s view: either $\sigma'' = x$, with x chosen uniformly at random as part of the query handling, or $\mathit{Evaluate}(s^*, \sigma'') = y^*$, with y^* not previously revealed to $\mathcal{A}$.

Let $\mathcal{Q}_G(M, i)$ be the set of queries algorithm $\mathcal{A}$ makes to its G oracle of the form $G(M, i, \cdot)$, i.e., the set of queries that can add a matching entry to G-table.

Then when algorithm $\mathcal{A}$ makes a query $H(M, i, \mu)$ with the one value of μ that triggers a consistency check, the probability that the consistency check will lead $\mathcal{B}$ to abort is at most $|\mathcal{Q}_G(M, i)|/|D|$, and the probability that $\mathcal{B}$ ever needs to abort for reason 1, regardless of how many H oracle queries $\mathcal{A}$ makes, is at most

$$\sum_{(M,i)} \frac{|\mathcal{Q}_G(M, i)|}{|D|} = \frac{1}{|D|} \sum_{(M,i)} |\mathcal{Q}_G(M, i)| \leq \frac{q_G}{|D|} \ .$$

(Crucially, for each M and i there is only one $H(M, i, \cdot)$ query that can trigger a consistency check, and so $\mathcal{Q}_G(M, i)$ is counted only once.)

We bound reason 2 similarly. Algorithm $\mathcal{B}$ is handling a G oracle query for $G(M, i, \sigma)$. For each M and i, there is exactly one value of σ that will trigger a consistency check, with $\mathcal{B}$ examining its H-table for entries matching $(M'' = M,\ i'' = i + 1,\ \mu'',\ z'',\ \mathit{good}'' = \texttt{false},\ \mathit{internal}'' = \texttt{false},\ x'')$.

There is exactly one value of μ'' in such an entry that would cause $\mathcal{B}$ to abort, and that this value depends on a value, z, that is chosen uniformly at random as part of the query handling and isn't in $\mathcal{A}$'s view.

Let $\mathcal{Q}_H(M, i)$ be the set of queries algorithm $\mathcal{A}$ makes to its H oracle of the form $G(M, i, \cdot)$, i.e., the set of queries that can add a matching entry to G-table. Then when algorithm $\mathcal{A}$ makes a query $G(M, i, \sigma)$ with the one value of σ that triggers a consistency check, the probability that the consistency check will lead $\mathcal{B}$ to abort is at most $|\mathcal{Q}_G(M, i + 1)|/|E|$, and the probability that $\mathcal{B}$ ever needs to abort for reason 1, regardless of how many G oracle queries $\mathcal{A}$ makes, is at most

$$\sum_{(M,i)} \frac{|\mathcal{Q}_H(M, i + 1)|}{|E|} = \frac{1}{|E|} \sum_{(M,i)} |\mathcal{Q}_H(M, i + 1)| \leq \frac{q_H}{|E|} \ .$$

To bound reason 3, we observe that if $\mathcal{A}$ did not query its H oracle at $H(M^*, i, \mu^*_{i-1})$ for some i then the value of $H(M^*, i, \mu^*_{i-1})$ is uniformly random and independent of $\mathcal{A}$'s view, and therefore so is the correct value for σ^*_i. We have already shown (in Section 4.2) that only one signature on M^* will verify as valid); it follows that only one value of σ^*_i will be valid as part of a signature. The value chosen (implicitly) by algorithm $\mathcal{A}$ is correct with probability $1/|D|$.

Similarly, to bound reason 4, we observe that if $\mathcal{A}$ did not query its G oracle at $H(M^*, i, \sigma^*_i)$ for some i then the value of $G(M^*, i, \sigma^*_i)$ is uniformly random and independent of $\mathcal{A}$'s view, and therefore so is the correct value for μ^*_i. We have already shown (in Section 4.2) that only one signature on M^* will verify as valid); it follows that only one value of μ^*_i will be valid as part of a signature. The value chosen (implicitly) by algorithm $\mathcal{A}$ is correct with probability $1/|E|$.

If $\mathcal{A}$ produces a valid forgery in an ϵ fraction of runs when run in the unforgeability experiment, it will produce a valid forgery without inducing a reason-1 through 4 abort in at least an $\epsilon - (q_G + 1)/|D| - (q_H + 1)/|E|$ fraction of runs under $\mathcal{B}$.

Now suppose that $\mathcal{A}$, running under $\mathcal{B}$, produces a valid forgery without inducing a reason-1 through 4 abort. We review the H-table and G-table maintained by $\mathcal{B}$. For each i, $1 \leq i \leq n$, we define the set $\mathcal{M}_i$ as follows: a message

$M \in \{0,1\}^*$ is included in $\mathcal{M}_i$ if there is an entry $(M' = M,\ i' = i,\ \mu',\ z',\ \mathit{good}' = \texttt{true},\ \mathit{internal}' = \texttt{false},\ x')$ in the H-table, for some μ', z', and x'. We further define the set $\mathcal{M}_{n+1}$ as follows: a message $M \in \{0,1\}^*$ is included in $\mathcal{M}_{n+1}$ if there is an entry $(M'' = M,\ i'' = n,\ \sigma'',\ z'',\ \mathit{good}'' = \texttt{true},\ \mathit{internal}'' = \texttt{false},\ \mu'')$ in the G-table, for some σ'', z'', and μ''.

Only an H oracle query made by $\mathcal{A}$ can cause a message M to be included in $\mathcal{M}_i$ for some $i \leq n$ (because $\mathit{internal}' = \texttt{false}$), and each H oracle query made by $\mathcal{A}$ can add only one entry to the H-table. We therefore know that $\sum_{i=1}^{n} |\mathcal{M}_i| \leq q_H$. Only a G oracle query made by $\mathcal{A}$ can cause a message M to be included in $\mathcal{M}_{n+1}$ (because $\mathit{internal}'' = \texttt{false}$), and each G oracle query made by $\mathcal{A}$ can add only one entry to the G-table.

In handling a G oracle query $G(M, i, \cdot)$ with $1 \leq i \leq n$, algorithm $\mathcal{B}$ will add an entry to the G-table with $\mathit{good} = \texttt{true}$ only if it finds a corresponding entry in the H-table with $M' = M$, $i' = i$, and $\mathit{good}' = \texttt{true}$. In handling an H oracle query $H(M, i, \cdot)$ with $2 \leq i \leq n$, algorithm $\mathcal{B}$ will add an entry to the H-table with $\mathit{good} = \texttt{true}$ only if it finds a corresponding entry in the G-table with $M' = M$, $i' = i - 1$, and $\mathit{good}' = \texttt{true}$. We therefore know that $\mathcal{M}_1 \supseteq \mathcal{M}_2 \supseteq \cdots \supseteq \mathcal{M}_n \supseteq \mathcal{M}_{n+1}$. Finally, since $\mathcal{B}$ produced a valid forgery on some message M^* but did not induce a reason-2 abort, we know that $M^* \in \mathcal{M}_i$ for all i and, in particular, that $M^* \in \mathcal{M}_{n+1}$. It follows that $|\mathcal{M}_{n+1}| > 0$.

We now restate two lemmas from Guo et al. [9].

Lemma 1 (Range Lemma [9]). *Let q and n be positive integers, and let $\mathcal{M}_1, \ldots, \mathcal{M}_{n+1}$ be sets satisfying $|\mathcal{M}_1| < q$, $|\mathcal{M}_{n+1}| > 0$, and $\mathcal{M}_1 \supseteq \mathcal{M}_2 \supseteq \cdots \supseteq \mathcal{M}_n \supseteq \mathcal{M}_{n+1}$. Then there exists an integer $i^* \in [1, n]$ such that*

$$|\mathcal{M}_{i^*}| < q^{(n+1-i^*)/n} \qquad \text{and} \qquad |\mathcal{M}_{i^*+1}| \geq q^{(n-i^*)/n}$$

Lemma 2 (Probability Lemma [9]). *Let q and n be positive integers, and let $\mathcal{M}_1, \ldots, \mathcal{M}_{n+1}$ be sets satisfying $|\mathcal{M}_1| < q$, $|\mathcal{M}_{n+1}| > 0$, and $\mathcal{M}_1 \supseteq \mathcal{M}_2 \supseteq \cdots \supseteq \mathcal{M}_n \supseteq \mathcal{M}_{n+1}$. Fix some arbitrary ordering on the elements of each set $\mathcal{M}_i$. Then if an integer c^* is chosen uniformly at random from the set $[1, n]$ and an integer k^* is chosen uniformly at random from the set $\left[1, q^{(n+1-c^*)/n}\right]$, the probability that the k^*th message in $\mathcal{M}_{c^*}$ is also in $\mathcal{M}_{c^*+1}$ is at least $1/\left(nq^{1/n}\right)$.*

For the proofs of these lemmas, see Guo et al. [9, Section 4]. Note that the Probability Lemma follows from the Range Lemma: We have $c^* = i^*$ with probability $1/n$, and conditioned on $c^* = i^*$ the probability that a message from $\mathcal{M}_{c^*}$ with index $k \in \left[1, q^{(n+1-c^*)/n}\right]$ is also in $\mathcal{M}_{c^*+1}$ is at least

$$\frac{|\mathcal{M}_{c^*+1}|}{\left|[1, q^{(n+1-c^*)/n}]\right|} \geq \frac{q^{(n-c^*)/n}}{q^{(n+1-c^*)/n}} = \frac{1}{q^{1/n}} \ .$$

Setting $q = q_H + 1$, we have already shown that the sets $\mathcal{M}_1, \ldots, \mathcal{M}_{n+1}$ defined by the hash table maintained by $\mathcal{B}$ satisfy the preconditions of the Probability Lemma in the case that $\mathcal{A}$ produces a valid forgery without inducing a reason-1 through 4 abort.

When algorithm $\mathcal{B}$ starts running, it chooses c^* uniformly at random from the set $[1, n]$ and k^* uniformly at random from the set $\left[1, q_S^{(n+1-c^*)/n}\right]$. It embeds the challenge in the k^*th entry in $\mathcal{M}_{c^*}$, where the arbitrary ordering on the elements of the sets $\mathcal{M}_{c^*}$ is the order of $\mathcal{A}$'s H oracle queries that cause elements to be added to $\mathcal{M}_{c^*}$.

In handling a signing query on a message M, algorithm $\mathcal{B}$ will find in its H-table an entry of the form $(M' = M,\ i' = i,\ \mu',\ z',\ \mathit{good}' = \texttt{true},\ \mathit{internal}' = \texttt{false},\ x')$ for each i, $1 \le i \le I$, and will find in its G-table an entry of the form $(M'' = M,\ i'' = i,\ \sigma'',\ z'',\ \mathit{good}'' = \texttt{true},\ \mathit{internal}'' = \texttt{false},\ \mu'')$ for each i, $1 \le i \le J$, where $I < n$ and J equals either $I-1$ or I. In the case that $I = J = n$, algorithm $\mathcal{B}$ has all the information it needs to answer the signing query. In any other case, it will make internal H oracle and G oracle queries in handling the signing query. These internal queries will add entries to the H-table and G-table, respectively, with $\mathit{good} = \texttt{true}$ and $\mathit{internal} = \texttt{true}$, which will prevent the later addition of entries with $\mathit{good} = \texttt{true}$ and $\mathit{internal} = \texttt{false}$.

In the notation we have just introduced, any message M submitted by $\mathcal{A}$ to its signing oracle will end up a member of $\mathcal{M}_1$ through $\mathcal{M}_I$ and, if $I = J = n$, of $\mathcal{M}_{n+1}$. A signing query on message M will induce a reason-5 abort if $J = I - 1$ and the H-table entry with $M' = M$ and $i' = I$ has $x' = \perp$. This can happen only if $\mathcal{B}$ chose to embed its inversion challenge in the response to a level-I H oracle query for the message M, but $\mathcal{A}$ did not then make a corresponding level-I G oracle query for the message M, revealing the solution to the inversion challenge. And, provided that $I < n$, if $\mathcal{A}$ did not make a level-I G oracle query for the message M it also did not make a level-$(I+1)$ H oracle query for the message M. But $\mathcal{B}$ embeds its inversion challenge in the k^*th entry in $\mathcal{M}_{c^*}$. Put another way: If the k^*th message in $\mathcal{M}_{c^*}$ is also in $\mathcal{M}_{c^*+1}$ then no signing oracle query will induce a reason-5 abort.

Finally, a reason 6 abort will occur if none of $\mathcal{A}$'s G oracle queries revealed to $\mathcal{B}$ the solution to the challenge it embedded in M, the k^*th message in $\mathcal{M}_{c^*}$. If $c^* = n$, then the presence of M in $\mathcal{M}_{n+1}$ guarantees that $\mathcal{A}$ made a level-n G oracle query for M, revealing the solution. If $c^* < n$, then the presence of M in $\mathcal{M}_{c^*+1}$ guarantees that $\mathcal{A}$ made a level-(c^*+1) H oracle query for M, which means it must also have made a level-c^* G oracle query for M, likewise revealing the solution. Put another way: If the k^*th message in $\mathcal{M}_{c^*}$ is also in $\mathcal{M}_{c^*+1}$, then $\mathcal{A}$ will not be forced into a reason-6 abort.

Under Guo et al.'s Probability Lemma, then, assuming that $\mathcal{A}$, running under $\mathcal{B}$, produces a valid forgery without inducing a reason-1 through 4 abort, no signing oracle query will induce a reason-5 abort and $\mathcal{B}$ will not be forced into a reason 6 abort with probability at least $1/\left(nq^{1/n}\right)$, where $q = q_H + 1$. Thus if $\mathcal{A}$ produces a valid forgery in an ϵ fraction of runs when run in the unforgeability experiment, $\mathcal{B}$ will use $\mathcal{A}$ to solve a one-wayness challenge in at least an $\left(\epsilon - (q_G+1)/|D| - (q_H+1)/|E|\right) \ / \ \left(n(q_H+1)^{1/n}\right)$ fraction of runs. Accounting for the running time incurred by $\mathcal{B}$ in answering $\mathcal{A}$'s oracle queries, we have proved the following theorem:

Theorem 1. *Let Π be a (t', ϵ')-one-way trapdoor permutation family over domain D. Then the short signature scheme on Π is $(t, q_H, q_G, q_S, \epsilon)$-secure against existential forgery under an adaptive chosen-message attack (in the random oracle model) for all t and ϵ satisfying*

$$\epsilon \geq n(q_H + 1)^{1/n}\epsilon' + \frac{q_G + 1}{|D|} + \frac{q_H + 1}{|E|} \qquad \textit{and} \qquad t \leq t' - O(q_H + q_G + nq_S) \ .$$

5 Instantiation with RSA

Lysyanskaya et al. explain how to use the RSA function to create a certified trapdoor permutation [12]. As usual, the permutation description consists of a modulus $N = pq$, where p and q are two large primes, along with e, relatively prime to $\varphi(N) = (p-1)(q-1)$. The trapdoor is $d = e^{-1} \mod \varphi(N)$. The domain D is $\mathbb{Z}/N\mathbb{Z}$, and the permutation is evaluated as

$$\pi \colon x \mapsto x^e \mod N \qquad \text{and} \qquad \pi^{-1} \colon y \mapsto y^d \mod N \ .$$

Two challenges remain.

First, we need $\pi(\cdot)$ to be a permutation of all of $\mathbb{Z}/N\mathbb{Z}$, even if N and e were maliciously generated. Lysyanskaya et al. extend $\pi(\cdot)$ as

$$\pi(x) = \begin{cases} x^e \bmod N & \text{if } x \text{ is relatively prime to } N \\ x & \text{otherwise} \end{cases}$$

and extend $\pi^{-1}(\cdot)$ similarly. If we knew that e is relatively prime to $\varphi(N)$, we would be done. Unfortunately, this property is not easy to verify given just N and e — indeed, it is intractable to verify when $e < N^{1/4}$, assuming the phi-hiding assumption holds, a fact used by Kakvi and Kiltz to build RSA full-domain hash signatures with tight security reduction [10]. Lysyanskaya et al., following Micali, Ohta, and Reyzin [13] and Cachin, Micali, and Stadler [5], propose to require that e prime and larger than N, properties that are easy to check and that are sufficient to guarantee that the extended $\pi(\cdot)$ above is a permutation. The downside to such large e is that applying $\pi(\cdot)$ takes time linear in $\log e$, which is why $e = 65537$ is frequently chosen in other applications of RSA.

Kakvi, Kiltz, and May observed that for prime e in the range $N^{\frac{1}{4}+\epsilon} < e < N$ it is possible to use Coppersmith's method to certify that e is relatively prime to $\varphi(N)$ in time $O(\epsilon^{-8} \log^2 N)$ [11]. Where verifiers are expected to process many signatures for each signing key they encounter, using smaller e and Kakvi-Kiltz-May certification instead of $e > N$ should reduce overall running time.

Second, we must pick the group operation for the domain $D = \mathbb{Z}/N\mathbb{Z}$. It cannot be $\times$, because, e.g., p has no multiplicative inverse modulo $N = pq$. Instead, following Lysyanskaya et al., we pick $+$ as our group operation.

Finally, we must select a group E. Here the only requirement is that $(q_H + 1)/|E|$ be negligible. It is sufficient for E to consist of bit strings of length twice the security parameter. At the 128-bit security level, for example, elements of E can be 256 bits long.

References

1. C. Bader, T. Jager, Y. Li, and S. Schäge. On the impossibility of tight cryptographic reductions. In M. Fischlin and J.-S. Coron, editors, *Proceedings of Eurocrypt 2016*, volume 9666 of *LNCS*, pages 273–304. Springer-Verlag, May 2016.
2. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. Denning, R. Pyle, R. Ganesan, R. Sandhu, and V. Ashby, editors, *Proceedings of CCS 1993*, pages 62–73. ACM Press, Nov. 1993.
3. M. Bellare and M. Yung. Certifying permutations: Non-interactive zero-knowledge based on any trapdoor permutation. *J. Cryptology*, 9(1):149–66, 1996.
4. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *J. Cryptology*, 17(4):297–319, Sept. 2004.
5. C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In J. Stern, editor, *Proceedings of Eurocrypt 1999*, volume 1592 of *LNCS*, pages 402–14. Springer-Verlag, May 1999.
6. J.-S. Coron. On the exact security of full domain hash. In M. Bellare, editor, *Proceedings of Crypto 2000*, volume 1880 of *LNCS*, pages 229–35. Springer-Verlag, Aug. 2000.
7. J.-S. Coron. Optimal security proofs for PSS and other signature schemes. In L. Knudsen, editor, *Proceedings of Eurocrypt 2002*, volume 2332 of *LNCS*, pages 272–87. Springer-Verlag, May 2002.
8. E.-J. Goh, S. Jarecki, J. Katz, and N. Wang. Efficient signature schemes with tight reductions to the diffie-hellman problems. *J. Cryptology*, 20(4):493–514, Oct. 2007.
9. F. Guo, R. Chen, W. Susilo, J. Lai, G. Yang, and Y. Mu. Optimal security reductions for unique signatures: Bypassing impossibilities with a counterexample. In J. Katz and H. Shacham, editors, *Proceedings of Crypto 2017*, volume 10402 of *LNCS*, pages 517–47. Springer-Verlag, Aug. 2017.
10. S. A. Kakvi and E. Kiltz. Optimal security proofs for full domain hash, revisited. In D. Pointcheval and T. Johansson, editors, *Proceedings of Eurocrypt 2012*, volume 7235 of *LNCS*, pages 537–53. Springer-Verlag, Apr. 2012.
11. S. A. Kakvi, E. Kiltz, and A. May. Certifying RSA. In X. Wang and K. Sako, editors, *Proceedings of Asiacrypt 2012*, volume 7658 of *LNCS*, pages 404–14. Springer-Verlag, Dec. 2012.
12. A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. Sequential aggregate signatures from trapdoor permutations. In C. Cachin and J. Camenisch, editors, *Proceedings of Eurocrypt 2004*, volume 3027 of *LNCS*, pages 74–90. Springer-Verlag, May 2004.
13. S. Micali, K. Ohta, and L. Reyzin. Provable-subgroup signatures. Unpublished manuscript, 1999.
14. G. Neven. Efficient sequential aggregate signed data. *IEEE Tran. Info. Th.*, 57(3):1803–15, Feb. 2011.
15. H. Orman and P. Hoffman. Determining strengths for public keys used for exchanging symmetric keys. RFC 3766, Apr. 2004.
16. D. Papadopoulos, D. Wessels, S. Huque, M. Naor, J. Včelák, L. Reyzin, and S. Goldberg. Making NSEC5 practical for DNSSEC. Cryptology ePrint Archive, Report 2017/099, 2017. `https://eprint.iacr.org/2017/099`.
17. E. R. Verheul. Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. *J. Cryptology*, 17(4):277–96, Sept. 2004.