A Unified Framework for Trapdoor-Permutation-Based Sequential Aggregate Signatures

Craig Gentry[1], Adam O'Neill[2], and Leonid Reyzin[3]

[1] Cryptography Research Group
Thomas J. Watson Research Center
Yorktown Heights, NY, USA
cbgentry@us.ibm.com
https://researcher.watson.ibm.com/researcher/view.php?person=us-cbgentry
[2] Department of Computer Science
Georgetown University
3700 Reservoir Road NW, Washington DC 20057, USA
adam@cs.georgetown.edu
http://cs.georgetown.edu/adam
[3] Department of Computer Science
Boston University
111 Cummington St., Boston, MA 02215, USA
reyzin@cs.bu.edu
http://cs.bu.edu/reyzin

Abstract. We give a framework for trapdoor-permutation-based sequential aggregate signatures (SAS) that unifies and simplifies prior work and leads to new results. The framework is based on *ideal ciphers over large domains*, which have recently been shown to be realizable in the random oracle model. The basic idea is to replace the random oracle in the full-domain-hash signature scheme with an ideal cipher. Each signer in sequence applies the ideal cipher, keyed by the message, to the output of the previous signer, and then inverts the trapdoor permutation on the result. We obtain different variants of the scheme by varying additional keying material in the ideal cipher and making different assumptions on the trapdoor permutation. In particular, we obtain the first scheme with lazy verification and signature size independent of the number of signers that does not rely on bilinear pairings.

Since existing proofs that ideal ciphers over large domains can be realized in the random oracle model are lossy, our schemes do not currently permit practical instantiation parameters at a reasonable security level, and thus we view our contribution as mainly conceptual. However, we are optimistic tighter proofs will be found, at least in our specific application.

Keywords. Aggregate signatures, trapdoor permutations, ideal cipher model.

1 Introduction

Aggregate signatures and their variants. Aggregate signature schemes (AS), introduced by Boneh et al. [6] (BGLS), allow n signatures on different messages produced by n different signers to be combined by any third party into a single short signature for greater efficiency, while maintaining the same security as n individual signatures. In this paper we are concerned with the more restricted *sequential* aggregate signatures (SAS), introduced by Lysyanskaya *et al.* (LMRS) [22] and further studied by [21, 5, 1, 24, 7, 17]. These schemes, while still maintaining the same security, require signers themselves to compute the aggregated signature in order, with the output of each signer (so-called "aggregate-so-far") used as input to the next during the signing process. This restriction turns out to be acceptable in several important applications of aggregate signatures, such as PKI certification chains and authenticated network routing protocols (*e.g.*, BGPsec).

TDP-based SAS. Existing SAS constructions are usually based on trapdoor permutations (TDPs) [22, 1, 24, 7] or bilinear pairings [21, 1, 5, 17]. In this paper, we focus on improving and simplifying TDP-based SAS schemes, which are all in the random oracle (RO) model. We describe existing constructions below, and illustrate them in Fig. 1.

The first TDP-based SAS scheme, by Lysyanskaya *et al.* [22] (LMRS), is very similar to the full-domain-hash (FDH) signature scheme of Bellare and Rogaway [2]. Recall that in FDH, the hash function is modeled as a random oracle whose range is equal to the domain of the TDP, and the signer simply hashes the message and inverts the TDP on the hash output. In LMRS, the signer exlcusive-ors the previous signer's output together with the hash of the message before inverting the TDP. This procedure enabled the verifier to verify in reverse order of signing, because exclusive-or could be undone to obtain the previous signer's (alleged) output.

Unfortunately, this very simple construction is not secure, and two additional checks are used in LMRS to achieve security: first, each signer must ensure that the public keys of all preceding signers are "certified" — *i.e.,* specify permutations; and second, each signer must verify the signature received from the previous signers before applying the signing operation. These two checks prevented fast signing; ideally, each signer would be able to sign independently of others, and verify when time permitted (this option is called "lazy verification" and was observed by Brogle *et al.* [7] to be crucial in authenticated network routing protocols).

In two successive works by Neven [24] and Brogle *et al.* [7] (BGR), these two additional checks were removed (permitting, in particular, lazy verification), but at a cost to simplicity and signature length. Neven's scheme eliminated the first check by introducing a Feistel-like structure with two hash functions, at the cost of lengthening the signature by a single hash value; BGR, building on top of Neven, eliminated the second check by lengthening the signature further by a

short per-signer value. These two schemes were complex and had subtle security proofs.

Our framework. We give a new framework for TDP-based SAS schemes, which unifies and simplifies prior work as well as leads to improved constructions. We observe that in all three prior TDP-based schemes, the central design question was how to process the aggregate-so-far together with the message before applying the hard direction of the TDP; in all three, it was accomplished using some combination of exclusive-or and random-oracle hash operations which were designed to ensure that the aggregate-so-far could be recovered during verification. In other words, achieving invertibility in order to enable verification was a major design constraint.

Our idea is to build invertibility explicitly into the scheme. In our scheme, pictured in Fig. 1, we process the aggregate-so-far via a public random permutation (modeled as an ideal cipher), keyed by the message. In other words, our schemes are in the ideal *ideal cipher* model, where algorithms have access to a family of random permutations and their inverses. This model is typically used for blockcipher-based constructions, where a blockcipher like AES is modeled as an ideal cipher. In our work, the domain of the ideal cipher is that of the trapdoor permutation, which is usually much larger than the block-length of a typical block cipher like AES. Fortunately, as shown by a series of works [10, 12–14] we can replace arbitrary-length ideal ciphers by using 8 rounds of Feistel network, and obtain the same security in the random oracle as in the ideal cipher model using indifferentiability arguments [23].

Our Results. Our framework not only simplifies prior work, but gives rise to more efficient aggregate signatures. Specifically, we obtain:

- A scheme that, like Neven's, does not require certified TDPs, but may permit shorter signatures than Neven's scheme, of length equal to the length of the TDP output.[4]
- A scheme that, like BGR, permits lazy verification, but retains constant-size signatures. This scheme is based on a stronger assumption of adaptive tag-based TDPs [20] instead of plain TDPs.

If one prefers to stay with plain TDPs, we also obtain a scheme that permits lazy verification and, like BGR, has signatures that grow with the number of signers, but still the signatures have the potential to be shorter than in BGR (with the same caveat as above). We do not compare computational costs of our scheme vs. Neven's and BGR's because the only difference is the (small) number of additional hashes, which is negligible compared to the cost of evaluating the TDP.

[4] For a comparison at the same security level, one must take into account losses in the security proofs. Unfortunately, the proofs of [10, 12–14] are lossy, so currently we have to use a much larger domain size of the ideal cipher than the TDP output. However, we are optimistic tighter proofs will be found; see open problems below.

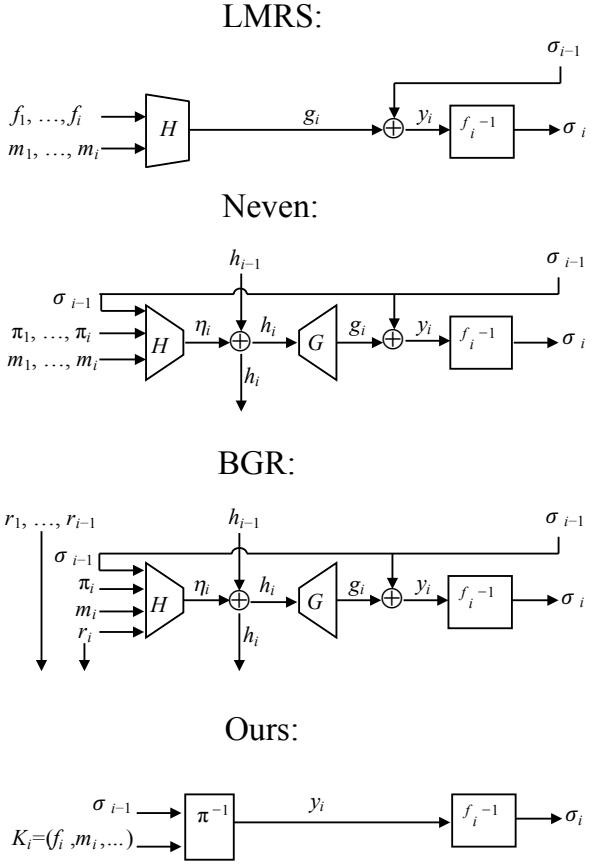

Fig. 1. Our framework, compared to LMRS, Neven, and BGR.

Main Technique: Chain-to-Zero Lemma. The security proofs for all our schemes are enabled by a lemma we prove about an ideal cipher keyed by descriptions of functions. We emphasize that the functions are unrelated to the ideal cipher itself, and that the cipher keyed by a function description results in permutation that is unrelated to the function. This lemma, which we call "Chain-to-Zero Lemma," states the following.

Let π_k denote the ideal cipher with key k. Recall that accessing π and π^{-1} requires querying an oracle. Let f and g denote functions with the same domain and range as π; the function descriptions will also be used as keys for π (again, we emphasize that the resulting permutations π_f and π_g have nothing to do with f and g as functions). Suppose for some a, $\pi_g(a) = b$, $f(b) = c$, and $\pi_f(c) = d$. We will say that a, b is *linked* to c, d. In our schemes, linking corresponds to consecutive steps of the verification algorithm.

A sequence of values in which each pair is linked to the next pair defines a *chain*. Signature verification will make sure that the last element of a chain is 0. The Chain-to-Zero Lemma says that if the last element of the chain is 0, then with overwhelming probability it was formed via queries to π^{-1} rather than to π. In our security proofs, this lemma means that we can program the relevant queries to π^{-1}, and therefore a forgery can be used to break the underlying TDP.

RSA-based instantiations. The schemes we obtain via our framework are proven secure under claw-freeness of the underlying TDP, or adaptive claw-freeness in the tag-based TDPs case. For plain-TDP-based schemes, this means that we can use RSA assuming standard one-wayness. For the tag-based TDP scheme, we we can use RSA under a stronger assumption called the instance-independent RSA assumption [25]. This instantiation hashes to the exponent (an idea originating from [18]), so verification is more expensive than for standard RSA.

Perspective and open problems. Compared to prior work, our framework pushes much of the complexity of security proofs to indifferentiability arguments that a Feistel network realizes an ideal cipher, and allows working with an ideal cipher as a clean abstraction. We point out two interesting directions for future work:

- Known proofs that a Feistel network is indifferentiable from an ideal cipher are lossy in the sense that the security guarantees obtained are weaker for a fixed domain size. We conjecture that a weaker property suffices to prove our Chain-to-Zero Lemma and can be realized via a tight proof. We leave proving or disproving this conjecture as an interesting direction for future work (and perhaps fewer Feistel rounds).
- The RSA-based instantiation of our tag-based TDP scheme has an expensive verification algorithm that performs a full exponentiation modulo N, and its security relies on a very strong assumption about RSA. It would be interesting to remove either of these drawbacks. We conjecture that one can actually prove a negative result here, namely that plain TDPs cannot be

used to realize constant-size lazy-verifying SAS schemes in the RO model, in a black-box way.

Finally, we mention that an open problem is removing the use of ROs in TDP-based SAS schemes, although our framework does not shed any light on this issue.

2 Preliminaries

2.1 Notation and Conventions

ALGORITHMS. If A is an algorithm then $y \leftarrow A(x_1, \ldots, x_n; r)$ means we run A on inputs $x_1, \ldots, x_n$ and coins r and denote the output by y. By $y \leftarrow\!\!\$\, A(x_1, \ldots, x_n)$ we denote the operation of picking r at random and letting $y \leftarrow A(x_1, \ldots, x_n; r)$. By $\Pr[P(x) : \ldots]$ we denote the probably that $P(x)$ holds after the elided experiment is executed. Unless otherwise indicated, an algorithm may be randomized. "PPT" stands for "probabilistic polynomial time" and "PT" stands for "polynomial time." The security parameter is denoted $k \in \mathbb{N}$. If we say that an algorithm is efficient we mean that it is PPT. All algorithms we consider are efficient unless indicated otherwise.

STRINGS AND VECTORS.We denote by $\{0,1\}^*$ the set of all (binary) strings, by $\{0,1\}^n$ the set of all strings of length $n \in N$, and by $\{0,1\}^{\geq n}$ the set of all strings of length *at least* $n \in \mathbb{N}$. If a, b are strings then $a \| b$ denotes an encoding from which a and b are uniquely recoverable. Vectors are denoted in boldface, for example $\mathbf{x}$. We sometimes use set notation with vectors, so that the notation $\mathbf{x} \leftarrow \mathbf{x} \cup \{x\}$ means that the next empty position in $\mathbf{x}$ is assigned x. If X is a random variable over some (finite) probability space then $\mathbf{E}[X]$ denotes its expectation.

TABLES. We use the term "table" to refer to an associative array implicitly initialized to empty. We use the pseudocode "Record $x = T[y]$ in the X-table" to mean that x is put at index y in table T. We use the pseudocode "X-table entry $x = T[y]$" to refer to x as the value at index y in table T.

SIMPLIFYING CONVENTIONS. We implicitly assume that an honestly generated secret key contains the matching public key. In experiments, we assume that an adversarially-provided public key can be parsed into the requisite form, and that if it contains a description of a function f then f is PT. This does *not* mean we assume public keys certified by a CA. Indeed, our requirement can be met by running f via inputting it and its input to some PT algorithm F, say universal machine that executes f on its input for a fixed amount of time; if f halts with some output then F outputs it as well, otherwise F outputs a default value. For simplicity, we also assume trapdoor permutations have domain $\{0,1\}^k$ but discuss RSA-based instantiations in Appendix A and Section 7.

2.2 Claw-Freeness

Claw-free trapdoor permutations. A *trapdoor permutation (TDP) generator* $\mathcal{F}$ on input 1^k outputs a pair (f, f^{-1}, g) describing permutations f, g on $\{0,1\}^k$, and f^{-1} describing the inverse of f. For a claw-finding algorithm C and every $k \in \mathbb{N}$, define its *CF-advantage* against $\mathcal{F}$ as

$$\mathbf{Adv}^{\mathrm{cf}}_{\mathcal{F},C}(k) = \Pr[f(x) = g(x') \,:\, (f, f^{-1}, g) \leftarrow\!\!\$\, \mathcal{F}\,;\, (x, x') \leftarrow\!\!\$\, C(f, g)]\ .$$

We say that $\mathcal{F}$ is a *claw-free* if $\mathbf{Adv}^{\mathrm{cf}}_{\mathcal{F},C}(\cdot)$ is negligible for every PPT C.

The permutation g is only used for security proofs. In our constructions, we will ignore g and write $(f, f^{-1}) \leftarrow\!\!\$\, \mathcal{F}(1^k)$, corresponding to the standard notion of trapdoor permutations.

(Adaptive) claw-free tag-based TDPs. A *tag-based trapdoor permutation (TB-TDP) generator* $\mathcal{F}_{tag}$ with tag-space $\{0,1\}^\tau$ on input 1^k outputs a pair $(f_{tag}, f^{-1}_{tag}, g_{tag})$ describing functions of two inputs: $t \in \{0,1\}^\tau$ (called the *tag*) and $x \in \{0,1\}^k$. For every tag $t \in \{0,1\}^\tau$, $f_{tag}(t,\cdot), g_{tag}(t,\cdot)$ are permutations and $f^{-1}_{tag}(t,\cdot)$ is the inverse of $f_{tag}(t,\cdot)$. For a claw-finding algorithm C and every $k \in \mathbb{N}$, define its *ACF-advantage* against $\mathcal{F}_{tag}$ as

$$\mathbf{Adv}^{\mathrm{acf}}_{\mathcal{F},C}(k) = \Pr[f(t,x) = g(t,x') \,:\, (f, f^{-1}, g) \leftarrow\!\!\$\, \mathcal{F}_{tag}(1^k)\,;\, t \leftarrow\!\!\$\, \{0,1\}^k\,;$$
$$(x, x') \leftarrow\!\!\$\, C^{f^{-1}(\cdot,\cdot)}(f, g, t)]$$

where we require that C does not make a query of the form $f^{-1}(t,\cdot)$ to its oracle. We say that $\mathcal{F}$ is *adaptive claw-free* if $\mathbf{Adv}^{\mathrm{cf}}_{\mathcal{F},C}(\cdot)$ is negligible for every such PPT C.

Intuitively, $\mathcal{F}$ is adaptive claw-free if it is hard to find a claw even given access to an inversion oracle for f that may be called on tags other than the challenge tag. The notion of adaptive claw-freeness is new to this work. It is an extension of the notion of adaptive one-wayness introduced by Kiltz et al. [20].

Instantiations. Dodis and Reyzin [16] show that any homomorphic or randomly self-reducible trapdoor permutation, in particular RSA [26] is claw-free (with a tight security reduction to one-wayness).

The notion of adaptive one-wayness for trapdoor permutations) (and more generally trapdoor functions) was introduced by Kiltz *et al.* [20]. They show that RSA gives rise to an adaptive one-way tag-based TDP under the instance-independent RSA assumption (II-RSA). In Appendix A we show that the same construction yields an adaptive claw-free tag-based TDP. In the construction, computing the forward direction is slower than for standard RSA, as it performs an exponentiation where the exponent is the length of the modulus rather than a small constant.

2.3 Random Oracle Model

In the *random oracle model* [2] all parties (all algorithms and adversaries) have oracle access to a function ("the random oracle") $H\colon \{0,1\}^* \to \{0,1\}^*$ where

for every $x \in \{0,1\}^*$ the value of $H(x)$ is chosen uniformly at random of some desired output length. By using standard domain separation, it is equivalent to give all parties oracle access to an unlimited number of independent random oracles $H_1, H_2, \ldots : \{0,1\}^* \to \{0,1\}^*$. It is a well-known heuristic proposed by [2] to instantiate these oracles in practice via functions constructed appropriately from a cryptographic hash function.

2.4 Ideal Cipher Model

In the version of the *ideal cipher model* [27] we consider, all parties (again, all algorithms and adversaries) have oracle access to two functions ("the ideal cipher"):

$$\pi\colon \{0,1\}^* \times \{0,1\}^{\geq k} \to \{0,1\}^{\geq k} \quad \text{and} \quad \pi^{-1}\colon \{0,1\}^* \times \{0,1\}^{\geq k} \to \{0,1\}^{\geq k} ,$$

where the first is such that for each $K \in \{0,1\}^*$ and each input length $n \geq k$, $\pi(K,\cdot)$ is an independent random permutation on $\{0,1\}^n$. The second is such that for each $K \in \{0,1\}^*$ and each input length $n \geq k$, $\pi^{-1}(K,\cdot)$ is the inverse of $\pi(K,\cdot)$ on $\{0,1\}^n$. Such a model has typically been used to analyze blockcipher-based constructions in the symmetric-key setting (see, *e.g.*, [4]), where the key length is fixed to the key length of the blockcipher and the input length is fixed to the block length.

Our constructions are in the public-key setting, the key length will be unbounded, and the input length will be at least as long as the input length of a trapdoor permutation (say 2048 bits in the case of RSA). To implement such an ideal cipher in the random oracle model, one can use a Feistel network. Indeed, in their seminal work, Coron *et al.* [10] show that a 14-round Feistel network, where the round functions are independent random oracles, is indifferentiable in the sense of Maurer *et al.* [23] from a random permutation, which can then be used to implement the ideal cipher in a straightforward way. Essentially, indifferentiability implies that any reduction using the random permutation can be translated to one in the random oracle model. A subsequent sequence of works [12–14] show that 8 rounds is sufficient; the minimal number of rounds is still open but known to be at least six. Unfortunately, none of these works are "tight" in the sense that the resulting reduction in the random oracle model will be very loose. An interesting question for future work is whether a weaker notion than indifferentiability from an ideal cipher suffices in our constructions.

3 Sequential Aggregate Signatures

Sequential aggregate signatures (SAS) were introduced by Lysyanskaya *et al.* [22] and were subsequently studied by [21, 5, 1, 24, 7, 17]. Following the work of Brogle *et al.* [7] and Fischlin *et al.* [17] (and in particular using terminology of the latter) we classify SAS schemes into two types: *general* and *history-free*. In a history-free scheme, the signing algorithm uses only on the current signer's secret key, the message, and the aggregate-so-far. In a general scheme, it may also use the public keys and messages of the *previous* signers.

3.1 The General Case

Syntax. A *(general) sequential aggregate signature* (SAS) scheme is a tuple $\mathsf{SAS} = (\mathsf{Kg}, \mathsf{AggSign}, \mathsf{AggVer})$ of algorithms defined as follows. The *key-generation* algorithm Kg on input 1^k outputs a public-key pk and matching secret-key sk. The *aggregate signing* algorithm $\mathsf{AggSign}$ on inputs a secret key sk_i, message m_i, aggregate-so-far σ_{i-1} and a list of pairs of public keys and messages $((pk_1, m_1), \ldots, (pk_{i-1}, m_{i-1}))$ outputs a new aggregate signature σ_i. The *aggregate verification* algorithm AggVer on inputs a list of public keys and messages $(pk_1, m_1), \ldots, (pk_i, m_i)$ and an aggregate signature σ_i outputs a bit.

Security. The security notion we use is the same as that in [24, 7] and originates from [1], who strengthen the original notion of [22] to allow repeating public keys (which they call "unrestricted" SAS). To a general SAS scheme SAS and a forger F we associate for every $k \in \mathbb{N}$ a *(general) SAS-unforgeability* experiment $\mathbf{Exp}^{\text{sas-uf}}_{\mathsf{SAS},F}(k)$ that runs in three phases:

- *Setup:* The experiment generates $(pk, sk) \leftarrow\!\!\$\ \mathsf{Kg}(1^k)$.
- *Attack:* Next, the experiment runs F on input pk with oracle access to $\mathsf{AggSign}(sk, \cdot, \cdot, \cdot)$.
- *Forgery:* Eventually, F halts with output parsed as $(pk_1, m_1), \ldots, (pk_n, m_n), \sigma$. The experiment outputs 1 iff: (1) $\mathsf{AggVer}((pk_1, m_1), \ldots, (pk_n, m_n), \sigma)$ outputs 1, (2) $pk_{i^*} = pk$ for some $1 \le i^* \le n$, and (3) F did not make an oracle query of the form $\mathsf{AggSign}(sk, m_{i^*}, ((pk_1, m_1), \ldots, (pk_{i^*-1}, m_{i^*-1})))$.

Define the *(general) SAS-unforgeability advantage* of F as

$$\mathbf{Adv}^{\text{sas-uf}}_{\mathsf{SAS},F}(k) = \Pr\left[\mathbf{Exp}^{\text{sas-uf}}_{\mathsf{SAS},F}(k) \text{ outputs } 1\right].$$

3.2 The History-Free Case

Syntax. A *history-free sequential aggregate signature* (HF-SAS) scheme is a tuple $\mathsf{HF\text{-}SAS} = (\mathsf{Kg}, \mathsf{AggSign}, \mathsf{AggVer})$ of algorithms defined as follows. The *key-generation* algorithm Kg on input 1^k outputs a public-key pk and matching secret-key sk. The *history-free aggregate signing* algorithm $\mathsf{AggSign}$ on inputs sk, m, σ' outputs a new aggregate signature σ. The *aggregate verification* algorithm AggVer on inputs a list of public key and messages $(pk_1, m_1), \ldots, (pk_i, m_i)$ and aggregate signature σ outputs a bit.

Security. Security in the history-free case is more restrictive on what is considered to be a forgery by the adversary than in the general case. In particular, we follow Brogle *et al.* [7] in our formulation of security here but leave investigation of a stronger security model due to Fischlin *et al.* [17] for furtur work. (As noted by [7], this strengthening is not needed in applications such as BGPsec.) To an HF-SAS scheme $\mathsf{HF\text{-}SAS}$ and a forger F be a forger we associate for every $k \in \mathbb{N}$ a *history-free SAS unforgeability* experiment $\mathbf{Exp}^{\text{hf-sas-uf}}_{\mathsf{SAS},F}(k)$ that runs in three phases:

- *Setup:* The experiment generates $(pk, sk) \leftarrow_{\$} \mathsf{Kg}(1^k)$.
- *Attack:* Next, the experiment runs F on input pk with oracle access to $\mathsf{AggSign}(sk, \cdot, \cdot)$.
- *Forgery:* Eventually, F halts with output parsed as $(pk_1, m_1), \ldots, (pk_n, m_n), \sigma$. The experiment outputs 1 iff: (1) $\mathsf{AggVer}((pk_1, m_1), \ldots, (pk_n, m_n), \sigma)$ outputs 1, (2) $pk_{i^*} = pk$ for some $1 \leq i^* \leq n$, and (3) F did not make an oracle query of the form $\mathsf{AggSign}(sk, m_{i^*}, \cdot)$.

Define the *history-free SAS-unforgeability advantage* of F as

$$\mathbf{Adv}^{\text{hf-sas-uf}}_{\mathsf{HF\text{-}SAS},F}(k) = \Pr\left[\mathbf{Exp}^{\text{hf-sas-uf}}_{\mathsf{HF\text{-}SAS},F}(k) \text{ outputs } 1\right].$$

3.3 Message Recovery

We also consider sequential aggregate signature schemes with *message recovery*, following [3, 24]. The goal is to save on bandwidth. Here we replace the verification algorithm by a recovery algorithm, which we view as taking as inputs a list of public keys and an aggregate signature and outputting either a list of messages, with the intended meaning that the verifier accepts each message as authentic under the respective public key, or $\perp$, indicating the aggregate signature is rejected.

4 Our Basic Schemes

We give three basic schemes: a general scheme (where the signing algorithm uses the public keys and messages of the previous signers in addition to the current signer's secret key and message), and two history-free schemes (where the signing algorithm uses only the current signer's secret key and message). In this section we only present the constructions and security theorems. We postpone the proofs since we later give our main lemma that unifies the proofs.

4.1 SAS_1: A General Scheme

Let $\mathcal{F}$ be a trapdoor permutation generator. Define $\mathsf{SAS}_1[\mathcal{F}] = (\mathsf{Kg}, \mathsf{AggSign}, \mathsf{AggVer})$ in the ideal cipher model with input length of π and π^{-1} fixed to $k \in \mathbb{N}$, and where $\mathsf{Kg}(1^k)$ outputs (f, f^{-1}) generated via $\mathcal{F}(1^k)$ and:

Alg $\mathsf{AggSign}(f_i^{-1}, m_i, \sigma_{i-1}, (f_1, m_1), \ldots, (f_{i-1}, m_{i-1}))$:
  //This is for the ith signer in the sequence:
  If $\mathsf{AggVer}((f_1, m_1), \ldots, (f_{i-1}, m_{i-1}), \sigma_{i-1})$ outputs 0 then
    Return $\perp$
  If $i = 1$ then $\sigma_{i-1} \leftarrow 0^k$
  $x_{i-1} \leftarrow \sigma_{i-1}$; $K_i \leftarrow f_1 \| m_1 \| \ldots \| f_i \| m_i$
  $y_i \leftarrow \pi^{-1}(K_i, x_{i-1})$; $x_i \leftarrow f_i^{-1}(y_i)$; $\sigma_i \leftarrow x_i$
  Return σ_i

Alg $\mathsf{AggVer}((f_1, m_1), \ldots, (f_n, m_n), \sigma)$:
  $x_n \leftarrow \sigma$
  For $i = n$ down to 1 do:
    $y_i \leftarrow f_i(x_i)$
    $K \leftarrow f_1 \| m_1 \| \ldots \| f_i \| m_i$
    $x_{i-1} \leftarrow \pi(K_i, y_i)$
  If $x_0 = 0^k$ then return 1
  Else return 0

Theorem 1. *Suppose $\mathcal{F}$ is claw-free. Then $\mathsf{SAS}_1[\mathcal{F}]$ is aggregate-unforgeable in the ideal cipher model. In particular, suppose there is a forger F against $\mathsf{SAS}_1[\mathcal{F}]$ making at most q_π ideal cipher queries and at most q_S signing queries. Then there is a claw-finding algorithm C against $\mathcal{F}$ such that for every $k \in \mathbb{N}$*

$$\mathbf{Adv}^{\text{sas-ufcma}}_{\mathsf{SAS}_1[\mathcal{F}],F}(k) \;\le\; \left(\frac{1}{1/(e(q_S+1)) - q_\pi/2^k}\right) \cdot \mathbf{Adv}^{\text{cf}}_{\mathcal{F},C}(k) + q_\pi^2/2^k \;.$$

The running-time of C is that of F plus minor bookkeeping.

4.2 SAS_2: A History-Free Scheme with Randomized Signing

Let $\mathcal{F}$ be a trapdoor permutation generator and $\rho = \rho(k)$ be an integer parameter. Define $\mathsf{SAS}_2[\mathcal{F}] = (\mathsf{Kg}, \mathsf{AggSign}, \mathsf{AggVer})$ where $\mathsf{Kg}(1^k)$ outputs (f, f^{-1}) generated via $\mathcal{F}(1^k)$ and:

Algorithm $\mathsf{AggSign}(f_i^{-1}, m_i, \sigma_{i-1})$:
//This is for the ith signer in the sequence:
If $i = 1$ then $x_0 \leftarrow 0^k$ and $r_0 \leftarrow \varepsilon$
Else $(x_{i-1}, \mathbf{r}_{i-1}) \leftarrow \sigma_{i-1}$
$r_i \xleftarrow{\$} \{0,1\}^\rho$; $K_i \leftarrow f_i \| m_i \| r_i$
$y_i \leftarrow \pi^{-1}(K_i, x_{i-1})$
$x_i \leftarrow f_i^{-1}(y_i)$; Append r_i to $\mathbf{r}_{i-1}$
$\sigma_i \leftarrow (x_i, \mathbf{r}_i)$
Return σ_i

Algorithm $\mathsf{AggVer}((f_1, m_1), \ldots, (f_n, m_n), \sigma)$:
$(\sigma_n, (r_1, \ldots, r_n)) \leftarrow \sigma$
$x_n \leftarrow \sigma_n$
For $i = n$ down to 1 do:
  $y_i \leftarrow f_i(x_i)$
  $K \leftarrow f_i \| m_i \| r_i$
  $x_{i-1} \leftarrow \pi(K_i, y_i)$
If $x_0 = 0^k$ then return 1
Else return 0

Theorem 2. *Suppose $\mathcal{F}$ is claw-free. Then $\mathsf{SAS}_2[\mathcal{F}]$ is aggregate-unforgeable in the ideal cipher model. In particular, suppose there is a forger F against $\mathsf{SAS}_2[\mathcal{F}]$ making at most q_H queries to H, at most q_π queries to the ideal cipher, and at most q_S signing queries. Then there is a claw-finding algorithm C against $\mathcal{F}$ such that for every $k \in \mathbb{N}$*

$$\mathbf{Adv}^{\text{hf-sas-ufcma}}_{\mathsf{SAS}_2[\mathcal{F}],F}(k) \;\le\; \frac{2^{\rho+k}}{(2^\rho - q_S^2)(2^k - q_\pi^2)} \cdot \mathbf{Adv}^{\text{cf}}_{\mathcal{F},C}(k) + q_\pi^2/2^k \;.$$

The running-time of C is that of F plus minor bookkeeping.

4.3 SAS_3: A History-Free Scheme with Deterministic Signing

To get intuition, we first sketch how to forge against $\mathsf{SAS}_2[\mathcal{F}]$ when randomness r_i is simply omitted. Let $K_i = f_i \| m_i$ be the ideal cipher key that the i-th signer "thinks" it is using. Let $K'_i = f_i \| m'_i$ be the ideal cipher key derived from a message m'_i that it will be duped into signing, and let x'_{i-1} be the real aggregate-so-far. We show how to derive a corresponding fake aggregate-so-far x_{i-1}. Let $y_i = \pi^{-1}(K_i, x_{i-1})$ be the value that the i-th signer will apply f_i^{-1} to. We want to make $y_i = \pi^{-1}(K'_i, x'_{i-1})$, so that the i-th signer is duped.

But this is easy: In order to force $y_i = \pi^{-1}(K'_i, x'_{i-1})$, we only have to choose $\pi^{-1}(K_i, x_{i-1}) = \pi^{-1}(K'_i, x'_{i-1})$ and therefore $x_{i-1} = \pi(K_i, \pi^{-1}(K'_i, x'_{i-1}))$. In essence, to solve this issue we make f_i depend on m_i as well.

OUR CONSTRUCTION. Let $\mathcal{F}_{tag}$ be a tag-based trapdoor permutation with tag-space $\{0,1\}^\tau$. Let $H\colon \{0,1\}^* \to \{0,1\}^\tau$ be a hash function modeled as a random oracle. Define $\mathsf{SAS}_3[\mathcal{F}] = (\mathsf{Kg}, \mathsf{AggSign}, \mathsf{AggVer})$ where $\mathsf{Kg}(1^k)$ outputs (f, f^{-1}) generated via $\mathcal{F}_{tag}(1^k)$ and:

Algorithm $\mathsf{AggSign}(f_i^{-1}, m_i, \sigma_{i-1})$:
//This is for the ith signer in the sequence:
$x_{i-1} \leftarrow \sigma_{i-1}$
If $i = 1$ then $\sigma_{i-1} \leftarrow 0^k$
$K_i \leftarrow f_i \| m_i$
$y_i \leftarrow \pi^{-1}(K_i, x_{i-1})$
$t_i \leftarrow H(f_i \| m_i)$
$x_i \leftarrow f_i^{-1}(t_i, y_i)$
Return $\sigma_i = x_i$

Algorithm $\mathsf{AggVer}((f_1, m_1), \ldots, (f_n, m_n), \sigma)$:
$x_n \leftarrow \sigma$
For $i = n$ down to 1 do:
$t_i \leftarrow H(f_i \| m_i)$
$y_i \leftarrow f_i(t_i x_i)$
$K \leftarrow f_i \| m_i \| r_i$
$x_{i-1} \leftarrow \pi(K_i, y_i)$
If $x_0 = 0^k$ then return 1
Else return 0

Theorem 3. *Suppose $\mathcal{F}_{tag}$ is adaptive claw-free. Then $\mathsf{SAS}_3[\mathcal{F}]$ is aggregate-unforgeable in the ideal cipher and random oracle models. In particular, suppose there is a forger F against $\mathsf{SAS}_3[\mathcal{F}]$ making at most q_H queries to the random oracle and at most q_π queries to the ideal cipher. Then there is a claw-finding algorithm C against $\mathcal{F}_{tag}$ such that for every $k \in \mathbb{N}$*

$$\mathbf{Adv}^{\text{hf-sas-ufcma}}_{\mathsf{SAS}_3[\mathcal{F}],F}(k) \;\le\; \frac{2^{k+\tau}}{(2^k - q_\pi^2)(2^\tau - q_H)} \cdot \mathbf{Adv}^{\text{acf}}_{\mathcal{F}_{tag},C}(k) + q_\pi^2/2^k \;.$$

The running-time of C is that of F plus minor bookkeeping.

5 The Chain-to-Zero Lemma

Here we give a main lemma that will unify security analyses of our schemes.

THE SETTING. Consider an adversary A executing in the ideal cipher model where the input and output length of the ideal cipher is fixed to $k \in \mathbb{N}$, and where a key of the ideal cipher also describes a function $f\colon \{0,1\}^k \to \{0,1\}^k$ unrelated to the function $\pi\colon f \times \{0,1\}^k \to \{0,1\}^k$. That is, A may submit a query to π of the form f, y to receive a random $x \in \{0,1\}^k$, or a query f, x to π^{-1} to receive a random $y \in \{0,1\}^k$.[5] For simplicity, we assume that A does not make the same query twice or ask redundant queries, *i.e.*, does not ask for $\pi^{-1}[f, x]$ if it already asked for $\pi[f, y]$ for some y and got x in response, or vice versa.

[5] In the game, we denote by "y" an input to π and by "x" its output for consistency with our constructions in Section 4.

LINKING. We say that π-table entry $x_1 = \pi[f_2, y_2]$ is *linked to* π-table entry $x_0 = \pi[f_1, y_1]$ if $f_1(x_1) = y_1$. For intuition, one can think of a π-table entry $x_0 = \pi[f_1, y_1]$ as indicating that f_1 applied to something (which in our constructions correspond to an aggregate-so-far) yielded y_1; this entry is linked if the "something" is also stored in the π-table. See Fig. 2 for a depiction. We inductively define a π-table entry $x = \pi[f, y]$ to be *chained to zero* if $x = 0^k$ or it is linked to an entry that is chained to zero. The *length of the chain* is defined naturally, where a chain consisting of a single entry $0^k = \pi[f_1, y_1]$ has length one. We say that π-table entry $x = \pi[f, y]$ is a *forward query* if it is defined upon A making a π query. Similarly, we say that π-table entry $x = \pi[f, y]$ is a *backward query* if it is defined upon A making a π^{-1} query.

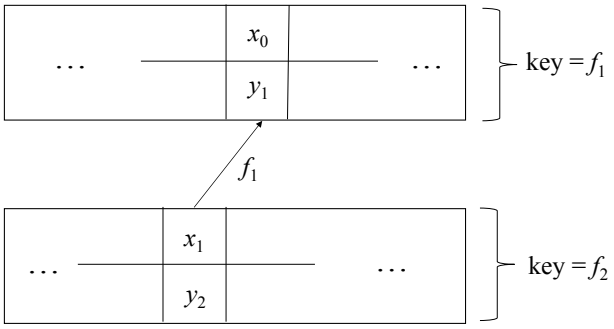

Fig. 2. A link between ideal cipher entries.

Lemma 4. (Chain-to-Zero Lemma) *Consider an execution A in which it makes at most q queries. Define BAD_π to be the event that some forward query gets chained to zero. Then* $\Pr[\,\mathsf{BAD}_\pi\,] \leq q^2/2^k$.

In the proof we will make use of the following claims.

Claim. Let $f\colon \{0,1\}^k \to \{0,1\}^k$. Consider choosing random $y_1, \ldots, y_q \in \{0,1\}^k$, and let $Y_{\max}$ be the random variable giving the maximum over i of the size of the pre-image set of $f^{-1}(y_i)$. Then $\mathbf{E}[Y_{\max}] = q$.

Proof. Let Y_i be the random variable giving the size of the pre-image set of $f^{-1}(y_i)$. We compute

$$\mathbf{E}[Y_{\max}] \;=\; \sum_{x=0}^{\infty} \Pr[Y_{\max} > x] \;\leq\; \sum_{x=0}^{\infty} \sum_{i=1}^{q} \Pr[Y_i > x] \;=\; \sum_{i=1}^{q} \mathbf{E}[Y_i] \;=\; q\,.$$

Above, for the first (in)equality we the fact that for a nonnegative integer-valued random variable X, $\mathbf{E}[X] = \sum_{x=0}^{\infty} \Pr[X > x]$. For the second inequality we use a union bound. For the last (in)equality we use that $E[Y_i] = 1$, because the expectation is simply the sum all pre-image set sizes divided by the total number of points. ∎

Now define Coll_1 to be the event that a forward query $x_i = \pi[f_{i+1}, y_{i+1}]$ is such that it is linked to some already existing backward query $x_{i-1} = \pi[f_i, y_i]$, and Coll_2 to be the event that a backward query $x_{i-1} = \pi[f_i, y_i]$ is such that it is linked to some already existing query $x_i = \pi[f_{i+1}, y_{i+1}]$ (either forward or backward). Define $\mathsf{Coll} = \mathsf{Coll}_1 \ \vee \ \mathsf{Coll}_2$.

Claim. In an execution A as above in which it makes at most q queries, we have $\Pr[\,\mathsf{Coll}\,] \leq q^2/2^k$.

Proof. We say that a forward query *collides* if satisfies the condition for Coll_1, and similarly for a backward query and Coll_2. After at most j backward queries have been made, define the random variable P_j to give the the maximum over all such queries of the size of the pre-image set $f^{-1}(y)$. We claim that after i queries, the probability a forward query collides is at most $i/2^k$. This is because for such a forward query $x = \pi[f, y]$, we have

$$\Pr[\,x = \pi[f,y] \text{ collides}\,] \ \leq \ \sum_{j=1}^{\infty} j \cdot \Pr[\,P_i = j\,] \cdot 2^{-k} \ = \ \mathbf{E}[P_i] \cdot 2^{-k} \ \leq \ i \cdot 2^{-k}\,,$$

where the last inequality is by the claim above.

Now if $x_{i-1} = \pi[f_i, y_i]$ is a backward query then y_i is random and independent, while for any existing query $x_i = \pi[f_{i+1}, y_{i+1}]$ we know $f_i(x_i)$ is already defined before y_i is chosen. So the probability $f_i(x_i) = y_i$ is 2^{-k}.

Hence, by a union bound the total probability of collision is at most $q^2/2^k$. ∎

We are now ready to prove our main lemma.

Proof. (of Lemma 4) By a conditioning argument, we have

$$\begin{aligned} \Pr[\,\mathsf{BAD}_\pi\,] \ &\leq \ \Pr\left[\,\mathsf{BAD}_\pi \mid \overline{\mathsf{Coll}}\,\right] + \Pr[\,\mathsf{Coll}\,] \\ &\leq \ \Pr\left[\,\mathsf{BAD}_\pi \mid \overline{\mathsf{Coll}}\,\right] + q^2/2^k \end{aligned}$$

using Claim 5.

Now if BAD_π occurs there are two possibilities, either some forward query $x = \pi[f, y]$ gets chained to zero by a chain of length $i = 1$, or it gets chained to zero by a chain of length $i > 1$. If $i = 1$ this would mean that $x = 0^k$. Since x is random and independent, the probability of this is 2^{-k}. Summing over all possible queries, the probability that any forward query gets chained to zero by a chain of length one is at most $q/2^k$.

Now suppose forward query $x_i = \pi[f_{i+1}, y_{i+1}]$ gets chained to zero by a chain of length $i > 1$. Then there are two possibilities: this query is chained to zero immediately when it is defined, or later.

The first possibility would require that there is a π-table entry $x_{i-1} = \pi[f_i, y_i]$ such that $f_i(x_i) = y_i$ and the entry is already chained to zero by a chain of length $i - 1$. By induction on i, $x_{i-1} = \pi[f_i, y_i]$ is a backward query, so it would cause a collision.

For the second possibility, consider a query that completes the chain from $x_{i-1} = \pi[f_i, y_i]$ to zero. At the time it is asked, all the other entries in the chain are already fixed. That query itself must be chained to zero via a chain of length j, for some $1 \le j \le i-1$, so let us denote it by $x_{j-1} = \pi[f_j, y_j]$. The query number $j+1$ in the chain, which we denote by $x_j = \pi[f_{j+1}, y_{j+1}]$, must be linked to query number j, i.e., it must hold that $f_j(x_j) = y_j$. Because query number $j-1$ must be chained to zero, again by (strong) induction on i it must be a backward query, so it would cause a collision.

This completes the proof. ∎

Remark 5. The Chain-to-Zero Lemma can be extended in the following way. Instead of functions $f\colon \{0,1\}^k \to \{0,1\}^k$ we allow functions $f\colon \{0,1\}^n \to \{0,1\}^n$, for any $n \ge k$, choose x and y in the game's pseudocode for answering A's queries of length n defined in the query, and define $x = \pi[f, y]$ to be *chained to zero* if $x = 0^k z^{n-k}$ for any $z \in \{0,1\}^{n-k}$, where n is the input length of f. The statement of the lemma remains unchanged.

6 Proofs for the Basic Schemes

Here we give security proofs of our basic schemes, using the Chain-to-Zero Lemma. To simplify the proofs, we assume that no query of forger F to the ideal cipher is asked twice (even in reverse direction) and that all queries needed in a signing query and for verifying the final forgery are already asked.

6.1 Proof of Theorem 1

We give a simpler proof that loses a factor q_π in the reduction rather than q_S; the improved reduction can be obtained via application of Coron's technique using biased coin flipping [9].

Claw finder. Claw-finding algorithm C is given in Fig. 3.

Analysis. Let's consider executions of the general SAS-unforgeability experiment with F and of the claw-finding experiment with C over a common set of random coin sequences, where the same coins are used for choices common across both experiments. Using the terminology of Section 5, in the execution of C in its claw-finding experiment let BAD_π be the event that any forward query is chained to zero and ABORT be the event that C aborts. Let FORGE be the event that F produces a valid forgery in its general SAS-unforgeability experiment. Then we have

$$\begin{aligned}\mathbf{Adv}^{\mathrm{cf}}_{\mathcal{F},C}(k) &\ge \Pr\left[\mathsf{FORGE} \wedge \overline{\mathsf{ABORT}} \wedge \overline{\mathsf{BAD}_\pi}\right] \\ &= \Pr\left[\mathsf{FORGE} \mid \overline{\mathsf{ABORT}} \wedge \overline{\mathsf{BAD}_\pi}\right] \cdot \Pr\left[\overline{\mathsf{ABORT}} \mid \overline{\mathsf{BAD}_\pi}\right] \cdot \\ &\quad \Pr\left[\overline{\mathsf{BAD}_\pi}\right] .\end{aligned}$$

The first inequality above is due to the fact that on coin sequences where C does not abort, the execution of F in its experiment and when run by C is identical. Hence, on such coin sequences F also forges in its execution by C.

Now by the Chain-to-Zero Lemma (Lemma 4), we have

$$\Pr\left[\,\overline{\mathsf{BAD}_\pi}\,\right] \;\geq\; 1 - q_\pi^2/2^k \;.$$

Next we claim that

$$\Pr\left[\,\overline{\mathsf{ABORT}} \mid \overline{\mathsf{BAD}_\pi}\,\right] \;\geq\; 1/q_\pi \;.$$

To see this, note that there are two places C could abort: answering a signing query, or after receiving the final forgery. In answering a signing query, we know that the aggregate-so-far must verify (otherwise C returns $\perp$), so π-table entry $x_{i-1} = \pi[f_1\|m_1\|\ldots\|f^*\|m_i, y_i]$ is chained to zero, and since we are conditioning on $\overline{\mathsf{BAD}_\pi}$ it must be a backward query. Similarly, upon receiving the F's final output, if it is a valid forgery then π-table entry $x^*_{i^*-1} = \pi[f_1^*\|m_1^*\|\ldots\|f^*\|m^*_{i^*}, y^*_{i^*}]$ must also be a backward query. So if C chooses ctr^* to be such that $x^*_{i^*-1} = \pi[f_1^*\|m_1^*\|\ldots\|f^*\|m^*_{i^*}, y^*_{i^*}]$ was defined on the ctr^*-th query, then C does not abort. This happens with probability at least $1/q_\pi$ since ctr^* is random and independent.

To complete the proof, we claim that

$$\Pr\left[\,\mathsf{FORGE} \mid \overline{\mathsf{ABORT}} \;\wedge\; \overline{\mathsf{BAD}_\pi}\,\right] \geq \mathbf{Adv}^{\text{sas-ufcma}}_{\mathsf{SAS}_1[\mathcal{F}],F}(k) - q_\pi^2/2^k \;.$$

To see this, first note that $\overline{\mathsf{ABORT}}$ is independent of FORGE because the random choices made by C in determining whether to abort in its claw-finding experiment do not affect whether F forges in its SAS-unforgeability experiment. Thus

$$\Pr\left[\,\mathsf{FORGE} \mid \overline{\mathsf{ABORT}} \;\wedge\; \overline{\mathsf{BAD}_\pi}\,\right] \;=\; \Pr\left[\,\mathsf{FORGE} \mid \overline{\mathsf{BAD}_\pi}\,\right] \;.$$

Now

$$\begin{aligned}
\Pr\left[\,\mathsf{FORGE} \mid \overline{\mathsf{BAD}_\pi}\,\right] \;&=\; \frac{\Pr\left[\,\mathsf{FORGE}\,\right] - \Pr\left[\,\mathsf{FORGE} \mid \mathsf{BAD}_\pi\,\right] \cdot \Pr\left[\,\mathsf{BAD}_\pi\,\right]}{\Pr\left[\,\overline{\mathsf{BAD}_\pi}\,\right]} \\
&\geq\; \Pr\left[\,\mathsf{FORGE}\,\right] - \Pr\left[\,\mathsf{BAD}_\pi\,\right] \\
&\geq\; \Pr\left[\,\mathsf{FORGE}\,\right] - q_\pi^2/2^k \\
&=\; \mathbf{Adv}^{\text{sas-ufcma}}_{\mathsf{SAS}_1[\mathcal{F}],F}(k) - q_\pi^2/2^k \;.
\end{aligned}$$

Combining the above, we have

$$\mathbf{Adv}^{\text{cf}}_{\mathcal{F},C}(k) \;\geq\; \left(\mathbf{Adv}^{\text{sas-ufcma}}_{\mathsf{SAS}_1[\mathcal{F}],F}(k) - q_\pi^2/2^k\right) \cdot \left(1/q_\pi - q_\pi/2^k\right)$$

and rearranging yields the theorem. ∎

Algorithm $C(f^*, g^*)$:
$ctr \leftarrow 0$; $ctr^* \leftarrow_{\$} \{1, \ldots, q_\pi\}$
Run F on input f^*, answering its queries as follows:
On π-query $f_1 \| m_1 \| \ldots \| f_i \| m_i, y$ do:
$x \leftarrow_{\$} \{0,1\}^k$
Record $\pi[f_1 \| m_1 \| \ldots \| f_i \| m_i, y] = x$ in the π-table ; Return x
On π^{-1}-query $f_1 \| m_1 \| \ldots \| f_i \| m_i, x$ do:
If $f_i = f^*$ then
$ctr \leftarrow ctr + 1$
If $ctr = ctr^*$ then
$x' \leftarrow_{\$} \{0,1\}^k$; $y \leftarrow g^*(x')$; Record $g^*[x'] = y$ in the g^*-table
Record $\pi[f_1 \| m_1 \| \ldots \| f_i \| m_i, y] = x$ in the π-table ; Return y
Else
$x' \leftarrow_{\$} \{0,1\}^k$; $y \leftarrow f^*(x')$; Record $f^*[x'] = y$ in the f^*-table
Record $\pi[f_1 \| m_1 \| \ldots \| f_i \| m_i, y] = x$ in the π-table ; Return y
On signing query $m_i, \sigma_{i-1}, (f_1, m_1), \ldots, (f_{i-1}, m_{i-1})$ do:
If $\mathsf{AggVer}((f_1, m_1), \ldots, (f_{i-1}, m_{i-1}), \sigma_{i-1})$ outputs 0 then return $\perp$
$x_{i-1} \leftarrow \sigma_{i-1}$; $y_i \leftarrow \pi^{-1}[f_1 \| m_1 \ldots f_{i-1} \| m_{i-1} \| f^* \| m_i, x_{i-1}]$
If y_i is not in the f^*-table then abort
Else let x_i be the index of y_i in the f^*-table
Return $\sigma = x_i$
Let $(f_1^*, m_1^*), \ldots, (f_n^*, m_n^*), \sigma^*$ be the output of F
If $\mathsf{AggVer}((f_1^*, m_1^*), \ldots, (f_n^*, m_n^*), \sigma^*)$ outputs 0 then return $\perp$
If there does not exist $1 \leq i^* \leq n$ such that $f_{i^*}^* = f^*$ then return $\perp$
$x_n^* \leftarrow \sigma^*$
For $i = n$ down to $i^* + 1$ do:
$y_i^* \leftarrow f_i^*(x_i^*)$
$x_{i-1}^* \leftarrow \pi[f_1^* \| m_1^* \| \ldots \| f_i^* \| m_i^*, y_i^*]$
$y_{i^*}^* \leftarrow f^*(x_{i^*}^*)$
If $y_{i^*}^*$ is not in the g^*-table then abort
Else let x'_{i^*} be the index of $y_{i^*}^*$ in the g^*-table
Return $(x_{i^*}^*, x'_{i^*})$

Fig. 3. Claw-finder C for the proof of Theorem 1.

6.2 Proof of Theorem 2

CLAW FINDER. Claw-finding algorithm C is given in Fig. 4.

ANALYSIS. Again, let's consider executions of the general SAS-unforgeability experiment with F and of the claw-finding experiment with C over a common set of random coin sequences with the same coins used for common choices across both experiments. Using the terminology of Section 5, in the execution of C in its claw-finding experiment let BAD_π be the event that any forward query gets chained to zero. Also in the execution of C in its experiment, let BAD_r be the event that π-table entry $\pi[f\|m\|r, y]$ defined when C answers signing query of F was previously defined. Let FORGE be the event that F produces a valid forgery in its experiment. We claim that

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{cf}}_{\mathcal{F},C}(k) &\geq \Pr\left[\,\mathsf{FORGE} \wedge \overline{\mathsf{BAD}_r} \wedge \overline{\mathsf{BAD}_\pi}\,\right] \\
&= \Pr\left[\,\mathsf{FORGE} \mid \overline{\mathsf{BAD}_r} \wedge \overline{\mathsf{BAD}_\pi}\,\right] \cdot \Pr\left[\,\overline{\mathsf{BAD}_r} \mid \overline{\mathsf{BAD}_\pi}\,\right] \cdot \Pr\left[\,\overline{\mathsf{BAD}_\pi}\,\right] \\
&\geq \Pr\left[\,\mathsf{FORGE} \mid \overline{\mathsf{BAD}_r} \wedge \overline{\mathsf{BAD}_\pi}\,\right] \cdot \Pr\left[\,\overline{\mathsf{BAD}_r} \mid \overline{\mathsf{BAD}_\pi}\,\right] \cdot \left(1 - q_\pi^2/2^k\right)
\end{aligned}$$

Above, the first inequality is because on a coin sequences on which F forges in its experiment and on which no π-table entry defined when C answers a signing query in its experiment was previously defined, the executions of both experiments are identical. Hence, on such coin sequences F also forges in its execution by C. Moreover, since the final output of F is a valid forgery, we know that π-table entry $x^*_{i^*-1} = \pi[f^*\|m^*_{i^*}\|r^*_{i^*}, y_{i^*}]$ is chained to zero. Since we are conditioning on $\overline{\mathsf{BAD}_\pi}$, the query on which the above π-table entry is defined must be a backward query, and since C populates the g^*-table on backwards queries, on such executions it can successfully find a claw. Finally, the last line is by the Chain-to-Zero Lemma.

Now we claim $\Pr\left[\,\mathsf{BAD}_r \mid \overline{\mathsf{BAD}_\pi}\,\right] \leq q_S^2/2^\rho$. This is because on each signing query r is chosen independently at random, in other words BAD_r and BAD_π are independent, and the probability that $x = \pi[(f, m, r), y]$ is already defined on a given signing query is at most $q_S/2^\rho$. Summing over all signing queries yields the claim.

Finally, we compute

$$\begin{aligned}
&\Pr\left[\,\mathsf{FORGE} \mid \overline{\mathsf{BAD}_r} \wedge \overline{\mathsf{BAD}_\pi}\,\right] = \\
&\frac{\Pr\left[\,\mathsf{FORGE}\,\right] - \Pr\left[\,\mathsf{FORGE} \mid \mathsf{BAD}_r \wedge \mathsf{BAD}_\pi\,\right] \cdot \Pr\left[\,\mathsf{BAD}_r \wedge \mathsf{BAD}_\pi\,\right]}{\Pr\left[\,\overline{\mathsf{BAD}_r} \wedge \overline{\mathsf{BAD}_\pi}\,\right]} \\
&\geq \Pr\left[\,\mathsf{FORGE}\,\right] - \Pr\left[\,\mathsf{BAD}_r \wedge \mathsf{BAD}_\pi\,\right] \\
&\geq \Pr\left[\,\mathsf{FORGE}\,\right] - \Pr\left[\,\mathsf{BAD}_\pi\,\right] \\
&= \mathbf{Adv}^{\text{sas-ufcma}}_{\mathsf{SAS}_1[\mathcal{F}],F}(k) - q_\pi^2/2^k\ .
\end{aligned}$$

where the last line uses the Chain-to-Zero Lemma. Combining terms yields the theorem. ∎

Algorithm $C(f^*, g^*)$:
Run F on input f^*, answering its queries as follows:
On π-query $f\|m\|r, y$ do:
$x \leftarrow\!\!\$\ \{0,1\}^k$
Record $\pi[f\|m\|r, y] = x$ in the π-table
Return x
On π^{-1} query $f\|m\|r, x$ do:
$x' \leftarrow\!\!\$\ \{0,1\}^k$; $y \leftarrow g^*(x')$
Record $g^*[x'] = y$ in the g^*-table
Record $\pi[f\|m\|r, y] = x$ in the π-table
Return y
On signing query m, σ do:
$(x_{i-1}, \mathbf{r}) \leftarrow \sigma$; $r \leftarrow\!\!\$\ \{0,1\}^\rho$
$x_i \leftarrow\!\!\$\ \{0,1\}^k$; $y_i \leftarrow f(x)$
Record $\pi[f\|m\|r, x_{i-1}] = y_i$ in the π-table
$\mathbf{r} \leftarrow \mathbf{r} \cup \{r\}$; $\sigma \leftarrow (x_i, \mathbf{r})$
Return σ
Let $(f_1^*, m_1^*), \ldots, (f_n^*, , m_n^*), \sigma^*$ be the output of F
If $\mathsf{AggVer}((f_1^*, m_1^*), \ldots, (f_n^*, m_n^*), \sigma^*)$ outputs 0 then return $\perp$
If there does not exist $1 \le i^* \le n$ such that $f_{i^*}^* = f^*$ then return $\perp$
$(x_n^*, \mathbf{r}^*) \leftarrow \sigma^*$
For $i = n$ down to $i^* + 1$ do:
$y_i^* \leftarrow f_i^*(x_i^*)$
$x_{i-1}^* \leftarrow \pi[f_i^* \| m_i^* \| r_i^*, y_i^*]$
$y_{i^*}^* \leftarrow f^*(x_{i^*}^*)$
Let x'_{i^*} be the index of y_{i^*} in the g^*-table
Return $(x_{i^*}^*, x'_{i^*})$

Fig. 4. Claw-finder C for the proof of Theorem 2.

6.3 Proof of Theorem 3

CLAW FINDER. Claw-finding algorithm C is given in Fig. 5.

ANALYSIS. Again, let's consider executions of the history-free SAS-unforgeability experiment with F and of the adaptive claw-finding experiment with C over a common set of random coin sequences, where the same coins are used choices common across both experiments. And, in the execution of C, let BAD_π be the event that any forward query is chained to zero. Let ABORT be the event that C aborts. Let FORGE be the event that F produces a valid forgery in its experiment. Then we have

$$\begin{aligned}\mathbf{Adv}^{\mathrm{acf}}_{\mathcal{F},C}(k) &\geq \Pr\left[\,\mathsf{FORGE} \wedge \overline{\mathsf{ABORT}} \wedge \overline{\mathsf{BAD}_\pi}\,\right] \\ &= \Pr\left[\,\mathsf{FORGE} \mid \overline{\mathsf{ABORT}} \wedge \overline{\mathsf{BAD}_\pi}\,\right] \cdot \Pr\left[\,\overline{\mathsf{ABORT}} \mid \overline{\mathsf{BAD}_\pi}\,\right] \cdot \\ &\quad \Pr\left[\,\overline{\mathsf{BAD}_\pi}\,\right] .\end{aligned}$$

The first inequality above is due to the fact that on coin sequences where C does not abort and no forward query made by F gets chained to zero, the execution of F in its experiment and when run by C is identical. Hence, on such coin sequences F also forges in its execution by C.

Now by the Chain-to-Zero Lemma (Lemma 4), we have

$$\Pr\left[\,\overline{\mathsf{BAD}_\pi}\,\right] \geq 1 - q_\pi^2/2^k .$$

Next we claim that

$$\Pr\left[\,\overline{\mathsf{ABORT}} \mid \overline{\mathsf{BAD}_\pi}\,\right] \geq 1/q_H \cdot (1 - q_H/2^\tau) .$$

To see this, note that there are three places C could abort: answering a hash query, answering a signing query, or after receiving the final forgery. Note that on each hash query where the "Else" is executed, we $t = t^*$ with probability $1/2^\tau$ since t and t^* are independent and random. Upon receiving the F's final output, if it is a valid forgery then π-table entry $x^*_{i^*-1} = \pi[f^* \| m^*_{i^*}, y^*_{i^*}]$ must be chained to zero and hence be a backward query. So if C chooses ctr^* to be such that $x^*_{i^*-1} = \pi[f^* \| m^*_{i^*}, y^*_{i^*}]$ was defined on the ctr^*-th query, then C does not abort. This happens with probability at least $1/q_H$ since ctr^* is random and independent.

To complete the proof, we claim that

$$\Pr\left[\,\mathsf{FORGE} \mid \overline{\mathsf{ABORT}} \wedge \overline{\mathsf{BAD}_\pi}\,\right] \geq \mathbf{Adv}^{\text{sas-ufcma}}_{\mathsf{SAS}_1[\mathcal{F}],F}(k) - q_\pi^2/2^k .$$

To see this, first note that $\overline{\mathsf{ABORT}}$ is independent of FORGE because the random choices made by C in determining whether to abort in its claw-finding experiment do not affect whether F forges in its SAS-unforgeability experiment. Thus

$$\Pr\left[\,\mathsf{FORGE} \mid \overline{\mathsf{ABORT}} \wedge \overline{\mathsf{BAD}_\pi}\,\right] = \Pr\left[\,\mathsf{FORGE} \mid \overline{\mathsf{BAD}_\pi}\,\right] .$$

Now

$$\begin{aligned}
\Pr\left[\,\mathsf{FORGE} \mid \overline{\mathsf{BAD}_\pi}\,\right] &= \frac{\Pr\left[\,\mathsf{FORGE}\,\right] - \Pr\left[\,\mathsf{FORGE} \mid \mathsf{BAD}_\pi\,\right] \cdot \Pr\left[\,\mathsf{BAD}_\pi\,\right]}{\Pr\left[\,\overline{\mathsf{BAD}_\pi}\,\right]} \\
&\geq \Pr\left[\,\mathsf{FORGE}\,\right] - \Pr\left[\,\mathsf{BAD}_\pi\,\right] \\
&\geq \Pr\left[\,\mathsf{FORGE}\,\right] - q_\pi^2/2^k \\
&= \mathbf{Adv}^{\text{sas-ufcma}}_{\mathsf{SAS}_1[\mathcal{F}],F}(k) - q_\pi^2/2^k
\end{aligned}$$

as claimed. Combining the above, we have

$$\mathbf{Adv}^{\text{cf}}_{\mathcal{F},C}(k) \;\geq\; \left(\mathbf{Adv}^{\text{sas-ufcma}}_{\mathsf{SAS}_1[\mathcal{F}],F}(k) - q_\pi^2/2^k\right) \cdot \left(1/q_H - q_\pi^2/2^k\right)$$

and rearranging yields the theorem. ∎

7 Extensions

We extend our basic schemes in a few ways. First, we add message recovery to them, so that we save on bandwidth. Second, we handle non-binary domains, as is needed for RSA-based instantiations.

7.1 Adding Message Recovery

To add message recovery to any of our schemes, the first signer can, instead of using the all-zeros string (of k-bits in length) as the first "aggregate-so-far," use n zero bits followed $n-k$ bits of the message for n equal to security parameter (here we abuse notation and use n as the security parameter, say 128, while k is the length of the modulus, say 2048). The security proofs are identical except that they use the extension of the Chain-to-Zero Lemma discussed in Remark 5. This gives us only security parameters number of bits of bandwidth overhead from the signature for sufficiently long messages. One issue is that the public keys of the signers still contribute to bandwidth overhead. It would be interesting for future work to treat message recovery for sequential aggregate signatures in the identity-based setting, which avoids public keys, as considered by [5].

7.2 Handling Non-Binary Domains

Our RSA-based instantiations in Appendix A have domain not $\{0,1\}^k$ but $\mathbb{Z}_N$ for per-signer N. The problem is that we may have a signer with modulus N_i and a subsequent signer with modulus N_{i+1} such that $N_{i+1} < N_i$. To handle this, there are two options. The first option is to append the fractional bit to the aggregate-so-far, so that the aggregate-so-far may grow by a bit per signer. This is quite modest growth, and in many applications such as S-BGP the number of signers is typically small. For highly bandwidth constrained applications, another option is to first convert the instantiation into one that does have a binary domain by using the technique of Hayashi, Okomoto, and Tanaka [19]. The idea is to exponentiate twice, reflecting the intermediate result about N. The downside is that this increases the cost of verification and signing by a factor of two.

Algorithm $C^{f^*_{inv}(\cdot,\cdot)}(f^*, g^*, t^*)$:
$ctr \leftarrow 0$; $ctr^* \leftarrow_\$ \{1, \ldots, q_H\}$
Run F on input f^* as follows:
On H-query $f \| m$ do:
If $f = f^*$ then
$ctr \leftarrow ctr + 1$
If $ctr = ctr^*$ then $t \leftarrow t^*$
Else $t \leftarrow_\$ \{0,1\}^k$; If $t = t^*$ then abort
Record $H[f\|m] = t$ in the H-table ; Return t
On π-query $f\|m, y$ do:
$x \leftarrow_\$ \{0,1\}^k$
Record $\pi[f\|m, y] = x$ in the π-table ; Return x
On π^{-1} query $f\|m, x$ do:
$x' \leftarrow_\$ \{0,1\}^k$; $y \leftarrow g^*(t^*, x')$
Record $g^*[t^*, x'] = y$ in the g^*-table
Record $\pi[f\|m, y] = x$ in the π-table ; Return y
On signing query m, σ do:
If $H[f^*\|m] = t^*$ then abort
$x_{i-1} \leftarrow \sigma$
$t_i \leftarrow H[f^*\|m]$; $y_i \leftarrow \pi[x_{i-1}]$; $x_i \leftarrow f^*_{inv}(t_i, y_i)$
Return $\sigma = x_i$
Let $(f^*_1, m^*_1), \ldots, (f^*_n, m^*_n), \sigma$ be the output of F
If $\mathsf{AggVer}((f^*_1, m^*_1), \ldots, (f^*_n, m^*_n), \sigma^*)$ outputs 0 then return $\perp$
If there does not exist $1 \le i^* \le n$ such that $f^*_{i^*} = f^*$ then return $\perp$
$x^*_n \leftarrow \sigma^*$
For $i = n$ down to $i^* + 1$ do:
$t^*_i \leftarrow H[f^*_i \| m^*_i]$
$y^*_i \leftarrow f^*_i(t^*_i, x^*_i)$
$x^*_{i-1} \leftarrow \pi[f^*_i \| m^*_i, y^*_i]$
$t^*_{i^*} \leftarrow H[f^*_{i^*} \| m^*_{i^*}]$
$y^*_{i^*} \leftarrow f^*(t^*_{i^*}, x^*_{i^*})$
If $y^*_{i^*}$ is not in the g^*-table then abort
Let x'_{i^*} be the index of $y^*_{i^*}$ in the g^*-table
Return $(x^*_{i^*}, x'_{i^*})$

Fig. 5. Adaptive Claw-finder C for the proof of Theorem 3.

Acknowledgements

We thank the anonymous reviewers of PKC 2018 for their helpful feedback, as well as Dana Dachman-Soled and Sharon Goldberg for helpful discussions. Adam O'Neill is supported in part by NSF grant 1650419. Leonid Reyzin is supported in part by NSF grant 1422965.

References

1. M. Bellare, C. Namprempre, and G. Neven. Unrestricted aggregate signatures. In L. Arge, C. Cachin, T. Jurdzinski, and A. Tarlecki, editors, *ICALP 2007*, volume 4596 of *LNCS*, pages 411–422. Springer, Heidelberg, July 2007.
2. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, Nov. 1993.
3. M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In U. M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 399–416. Springer, Heidelberg, May 1996.
4. J. Black, P. Rogaway, and T. Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 320–335. Springer, Heidelberg, Aug. 2002.
5. A. Boldyreva, C. Gentry, A. O'Neill, and D. H. Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *ACM CCS 07*, pages 276–285. ACM Press, Oct. 2007.
6. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, Heidelberg, May 2003.
7. K. Brogle, S. Goldberg, and L. Reyzin. Sequential aggregate signatures with lazy verification from trapdoor permutations - (extended abstract). In X. Wang and K. Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 644–662. Springer, Heidelberg, Dec. 2012.
8. B. Chevallier-Mames and M. Joye. Chosen-ciphertext secure RSA-type cryptosystems. In J. Pieprzyk and F. Zhang, editors, *ProvSec 2009*, volume 5848 of *LNCS*, pages 32–46. Springer, Heidelberg, Nov. 2009.
9. J.-S. Coron. On the exact security of full domain hash. In M. Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 229–235. Springer, Heidelberg, Aug. 2000.
10. J.-S. Coron, T. Holenstein, R. Künzler, J. Patarin, Y. Seurin, and S. Tessaro. How to build an ideal cipher: The indifferentiability of the Feistel construction. *Journal of Cryptology*, 29(1):61–114, Jan. 2016.
11. J.-S. Coron and D. Naccache. Security analysis of the Gennaro-Halevi-Rabin signature scheme. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 91–101. Springer, Heidelberg, May 2000.
12. D. Dachman-Soled, J. Katz, and A. Thiruvengadam. 10-round feistel is indifferentiable from an ideal cipher. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 649–678. Springer, Heidelberg, May 2016.

13. Y. Dai and J. Steinberger. Indifferentiability of 10-round Feistel networks. Cryptology ePrint Archive, Report 2015/874, 2015. http://eprint.iacr.org/2015/874.
14. Y. Dai and J. P. Steinberger. Indifferentiability of 8-round feistel networks. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 95–120. Springer, Heidelberg, Aug. 2016.
15. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
16. Y. Dodis and L. Reyzin. On the power of claw-free permutations. In S. Cimato, C. Galdi, and G. Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 55–73. Springer, Heidelberg, Sept. 2003.
17. M. Fischlin, A. Lehmann, and D. Schröder. History-free sequential aggregate signatures. In I. Visconti and R. D. Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 113–130. Springer, Heidelberg, Sept. 2012.
18. R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In J. Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 123–139. Springer, Heidelberg, May 1999.
19. R. Hayashi, T. Okamoto, and K. Tanaka. An RSA family of trap-door permutations with a common domain and its applications. In F. Bao, R. Deng, and J. Zhou, editors, *PKC 2004*, volume 2947 of *LNCS*, pages 291–304. Springer, Heidelberg, Mar. 2004.
20. E. Kiltz, P. Mohassel, and A. O'Neill. Adaptive trapdoor functions and chosen-ciphertext security. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 673–692. Springer, Heidelberg, May 2010.
21. S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. Sequential aggregate signatures and multisignatures without random oracles. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 465–485. Springer, Heidelberg, May / June 2006.
22. A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. Sequential aggregate signatures from trapdoor permutations. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 74–90. Springer, Heidelberg, May 2004.
23. U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In M. Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Heidelberg, Feb. 2004.
24. G. Neven. Efficient sequential aggregate signed data. In N. P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 52–69. Springer, Heidelberg, Apr. 2008.
25. P. Paillier and J. L. Villar. Trading one-wayness against chosen-ciphertext security in factoring-based encryption. In X. Lai and K. Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 252–266. Springer, Heidelberg, Dec. 2006.
26. R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.
27. C. E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949.

A RSA-Based Instantiations

We first define a general parameter generation algorithm used in our constructions. An RSA [26] parameter generation algorithm is an algorithm RSAGen that on input 1^k outputs (N, p, q, e, d) where $N = pq$, p and q are $m/2$-bit primes for some $m = m(k)$, and $ed = 1 \bmod \phi(N)$.

RSA TRAPDOOR PERMUTATION. An RSA trapdoor permutation generator $\mathcal{F}_{rsa}$ on input 1^k returns $f_{rsa} = (N, e), f_{rsa}^{-1} = (N, d)$ where $(N, e, d, p, q) \leftarrow\!\!\$\ \mathsf{RSAGen}(1^k)$. On input $x \in \mathbb{Z}_N^*$ algorithm f_{rsa} outputs $x^e \bmod N$ and on input $y \in \mathbb{Z}_N^*$ algorithm f_{rsa}^{-1} outputs $y^d \bmod N$. Dodis and Reyzin [16] show that the RSA trapdoor permutation generator is claw-free under the standard assumption it is one-way.

RSA TAG-BASED TRAPDOOR PERMUTATION. An RSA tag-based trapdoor permutation generator from Kiltz *et al.* [20] works as follows. Let $H\colon \{0,1\}^\tau \to \{0,1\}^\eta$ for some $\eta \in \mathbb{N}$ be a hash function. Define the tag-based trapdoor permutation generator $\mathcal{F}_{rsa\text{-}tag}$ with tag-space $\{0,1\}^\tau$ that on input 1^k outputs

$$f_{rsa\text{-}tag} = N\ ;\ f_{rsa\text{-}tag}^{-1} = (p, q)$$

for where $(N, p, q, e, d) \leftarrow\!\!\$\ \mathsf{RSAGen}$. On inputs $t \in \{0,1\}^\tau, x \in \mathbb{Z}_N^*$, algorithm $f_{rsa\text{-}tag}$ outputs $x^{H(t)} \bmod N$. On inputs $t \in \{0,1\}^\tau, y \in \mathbb{Z}_N^*$, algorithm $f_{rsa\text{-}tag}^{-1}$ computes $d \leftarrow H(t)^{-1} \bmod \phi(N)$ and outputs $y^d \bmod N$. Kiltz *et al.* [20] show that this tag-based trapdoor permutation generator is adaptive one-way assuming the *instance-independent RSA assumption* [25, 8, 20] holds and H is *division-intractable* [18]. This is plausible if $\eta = m$ (the modulus length) [11]. The same proof strategy of Dodis and Reyzin [16] works in the adaptive case and we thus obtain that this tag-based trapdoor permutation generator is adaptive claw-free under the same assumptions.