Work partly done while at Katholieke Universiteit Leuven, Belgium. An extended abstract of this paper appeared in Nigel Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 52-69, Springer-Verlag, 2008 [Nev08]. This is the full version.

Efficient Sequential Aggregate Signed Data

Gregory Neven

IBM Research – Zurich
Säumerstrasse 4, 8803 Rüschlikon, Switzerland
nev@zurich.ibm.com
http://www.neven.org

Abstract

We generalize the concept of sequential aggregate signatures (SAS), proposed by Lysyanskaya, Micali, Reyzin, and Shacham at Eurocrypt 2004, to a new primitive called *sequential aggregate signed data* (SASD) that tries to minimize the total amount of transmitted data, rather than just signature length. We present SAS and SASD schemes that offer numerous advantages over the scheme of Lysyanskaya et al. Most importantly, our schemes can be instantiated with *uncertified* claw-free permutations, thereby allowing implementations based on low-exponent RSA and factoring, and drastically reducing signing and verification costs. Our schemes support aggregation of signatures under keys of different lengths, and the SASD scheme even has as little as 160 bits of bandwidth overhead. Finally, we present a multi-signed data scheme that, when compared to the state-of-the-art multi-signature schemes, is the first scheme with non-interactive signature generation not based on pairings. All of our constructions are proved secure in the random oracle model based on families of claw-free permutations.

Keywords: Provable security, sequential aggregate signatures, message recovery.

1 Introduction

Aggregate signatures (AS) [BGLS03] allow any third party to compress individual signatures $\sigma_1, \ldots, \sigma_n$ by n different signers on n different messages into an aggregate signature σ of size considerably shorter than the sum of the sizes of the individual signatures, preferably independent of the number of signatures n. Sequential aggregate signatures (SAS) [LMRS04] are a slightly restricted variant where the signers have to be organized in a sequence, each taking turns in adding their signature share onto the aggregate. Example applications of (S)AS schemes include secure routing protocols [KLS00], where routers authenticate paths in the network, and certificate chains in hierarchical public-key infrastructures, where certificate authorities (CA) authenticate public keys of lower-level CAs. Another important application area is that of battery-powered devices such as cell phones, PDAs, and wireless sensors that communicate over energy-consuming wireless channels.

Drawbacks of existing schemes. In the public-key (i.e., non-identity-based) setting, only three (S)AS schemes are presently known: the pairing-based $\mathcal{BGLS}$ [BGLS03] and $\mathcal{LOSSW}$ [LOS+06] schemes, and the $\mathcal{LMRS}$ [LMRS04] scheme based on families of certified [BY96] trapdoor permutations, but that with some tricks can be instantiated with RSA. All three schemes have some drawbacks though.

Pairings were rather recently introduced to cryptography, and for the time being do not yet enjoy the same level of support in terms of standardization and implementations as for example RSA. The main disadvantage of the $\mathcal{LMRS}$ scheme on the other hand is that one of the tricks needed to turn RSA into a certified permutation is to use a verification exponent $e > N$.[1] This has a dramatic effect on the computational efficiency of signing and verification, because both require n long-exponent exponentiations for an aggregate signature containing n signatures.

Comparing this to pairing-based alternatives, the $\mathcal{BGLS}$ scheme also has rather expensive verification (n pairing computations), but at least has cheap signing (a single exponentiation). The $\mathcal{LOSSW}$ scheme has quite cheap signing and verification (two pairings and $160n$ multiplications), albeit at the price of only being secure in the weaker *knowledge of secret key* (KOSK) model that requires signers to hand over (or at least prove knowledge of) their secret keys to a trusted CA. Both pairing-based schemes have shorter signatures than the $\mathcal{LMRS}$ scheme: for a typical security level of 80 bits, the $\mathcal{BGLS}$ and $\mathcal{LOSSW}$ schemes have 160 and 320-bit signatures, respectively, versus 1024 bits for $\mathcal{LMRS}$.

Finally, none of the existing schemes give the signers much freedom in choosing their own key sizes. This is particularly important for the certificate chain application, where a top-level CA probably wants higher-grade security than a private end-user. The pairing-based schemes require all signers to use the same elliptic-curve groups, so the signers have no influence over their key sizes whatsoever. The $\mathcal{LMRS}$ scheme offers a limited amount of freedom, but requires that signers within a sequence are arranged according to *increasing* key size, which is exactly the opposite of what is needed for certificate chains.

Our contributions. We first observe that if one is truly concerned about saving bandwidth, then focusing solely on signature length is a bit arbitrary. Indeed, what really matters is the total amount of transmitted data, which contains messages, signatures, and in many applications the signers' public keys. (In fact, replacing the latter with shorter identity strings is the main motivation for identity-based aggregate signatures [GR06, BGOY07].) We therefore state our results in terms of a new, generalized primitive that we call *sequential aggregate signed data* (SASD). The verification algorithm takes as only input the signed data Σ, and outputs vectors of public keys $\vec{pk} = (pk_1, \ldots, pk_n)$ and messages $\vec{M} = (M_1, \ldots, M_n)$ to indicate that Σ correctly authenticates M_i under pk_i for $1 \leq i \leq n$, or outputs $(\perp, \perp)$ to reject. The goal of the scheme is to keep the *net bandwidth overhead* to a minimum, i.e., the difference between the length of the signed data $|\Sigma|$ and that of the useful content $\sum_{i=1}^{n} |M_i|$.

Next, we present our main construction, the $\mathcal{SASD}$ scheme, based on families of trapdoor permutations in the random oracle model. Its main advantage over the $\mathcal{LMRS}$ scheme is that it does *not* require the permutations to be certified, thereby allowing much more efficient instantiations from low-exponent RSA and the first instantiation ever from factoring. The construction itself can be seen as combining ideas from the $\mathcal{LMRS}$ scheme and the PSS-R signature scheme with message recovery [BR96]; the main technical contribution, we think, lies in the security

[1] Alternatively to choosing $e > N$, one could let each signer append to his public key a non-interactive zero-knowledge (NIZK) proof [BFM88] that $\gcd(e, \varphi(N)) = 1$. However, whether general NIZK proofs or special-purpose techniques [CM99, CPP07] are used, this invariably leads to a blowup in public key size and verification time, annihilating the gains of using aggregate signatures.

Scheme	Type	KOSK	RO	Prob	Overhead	Sign	Vf
$\mathcal{BGLS}$ [BGLS03]	AS	N	Y	P	k_p	1E	nP
$\mathcal{LOSSW}$ [LOS+06]	SAS	Y	N	P	$2k_\mathrm{p}$	$2\mathrm{P} + n\ell\mathrm{M}$	$2\mathrm{P} + n\ell\mathrm{M}$
$\mathcal{LMRS}$ [LMRS04]	SAS	N	Y	R	k_f	nE	nE
$\mathcal{SASD}$	SASD	N	Y	R,F	$[\ell, k_\mathrm{f} + \ell + n]$	$1\mathrm{E} + 2n\mathrm{M}$	$2n$M
$\mathcal{SAS}$	SAS	N	Y	R,F	$k_\mathrm{f} + \ell + n$	$1\mathrm{E} + 2n\mathrm{M}$	$2n$M
$\mathcal{Bol}$ [Bol03]	MS	Y	Y	P	k_p	1E	$2\mathrm{P} + n\mathrm{M}$
$\mathcal{LOSSW}$ [LOS+06]	MS	Y	N	P	$2k_\mathrm{p}$	$2\mathrm{E} + \ell\mathrm{M}$	$2\mathrm{P} + (\ell + n)\mathrm{M}$
$\mathcal{MSD}$	MSD	N	Y	R,F	$[\ell + n, nk_\mathrm{f} + \ell]$	1E	$2n$M

Table 1: Comparison of existing aggregate signature (AS), sequential aggregate signature (SAS), sequential aggregate signed data (SASD), multi-signature (MS), and multi-signed data (MSD) schemes. For each scheme we display whether its security relies on the knowledge of secret key (KOSK) or random oracle (RO) assumptions, on which number-theoretic problems it can be based (P for pairings, R for RSA, F for factoring), the net bandwidth overhead in bits, the cost of signing, and the cost of verification. Only the predominant terms are displayed in efficiency measures. Symbols used are security parameters k_p, k_f, ℓ for pairings, factoring, and collision-resistance (typical values are $k_\mathrm{p} = \ell = 160$, $k_\mathrm{f} = 1024$); n for the number of signers in an aggregation; P for a pairing operation; E for a (multi-)exponentiation; and M for a multiplication. The overhead of the $\mathcal{SASD}$ and $\mathcal{MSD}$ schemes is displayed as an interval, as their overhead depends on the length of the messages being signed. More details on efficiency are provided in the sections where the schemes are introduced.

---

proof, which requires complex "query bookkeeping" for the simulation to go through. The impact on efficiency is spectacular (see Table 1): verification takes a mere $2n$ multiplications, signing takes one exponentiation and $2n$ multiplications, and bandwidth overhead can be as low as 160 bits — something that until now seemed the exclusive privilege of pairing-based schemes. Moreover, the scheme allows signers to mix-and-match different key sizes at will, allowing much more flexibility for use in real applications.

There is a small caveat that our scheme only achieves its optimal bandwidth overhead for sufficiently long signed messages. Roughly, if the n signers in the aggregation have key sizes $k_1, \ldots, k_n$, then we need that $|M_i| \geq k_i - k_{i-1}$. To show that our efficiency gains are not only due to our generalization of the SAS primitive however, we also present a "purebred" SAS scheme that in most cases will have a larger bandwidth overhead than the $\mathcal{SASD}$ scheme, about $\max(k_1, \ldots, k_n) + 160$ bits to be exact, but that otherwise shares all the advantages in efficiency and flexibility of the $\mathcal{SASD}$ scheme.

Multi-signatures. A multi-signature (MS) scheme [IN83] is the natural equivalent of a (S)AS scheme where all signers authenticate the same message. The current state-of-the-art schemes based on RSA or factoring [BN06] have interactive signature generation; those based on pairings [Bol03, LOS+06] are only secure in the KOSK setting. The $\mathcal{BGLS}$ scheme could be seen as a MS scheme (taking into account the issues [BNN07] that arise when signing the same message), but has significantly less efficient verification.

Analogously to what we did for SASD schemes, we generalize the concept of MS schemes to *multi-signed data* (MSD) schemes. We present the $\mathcal{MSD}$ scheme that is the first RSA and factoring-based scheme with non-interactive signature generation, and that is the first efficient non-interactive scheme secure in the plain public-key setting, i.e. without making the KOSK assumption. Unlike the $\mathcal{SASD}$ scheme however, the bandwidth gains here are solely due to message recovery effects, and disappear completely when very short messages are being signed.

2 Sequential Aggregate Signed Data

NOTATION. If $k \in \mathbb{N}$, then 0^k is the bit string containing k zeroes, and $\{0,1\}^k$ is the set of all k-bit strings. If x, y are bit strings, then $|x|$ denotes the length (in bits) of x, $x\|y$ denotes a bit string from which x and y can be unambiguously reconstructed, and (x, y) denotes a tuple of bit strings. If $k \in \mathbb{N}$, S is a set, and $y \in S$, then $\vec{x} = (x_1, \ldots, x_k) \in S^k$ is a k-dimensional vector, $\vec{x}|y$ is the $(k+1)$-dimensional vector $(x_1, \ldots, x_k, y)$, and $\vec{x}|_i = (x_1, \ldots, x_i)$. Let ε and $\vec{\varepsilon}$ denote the empty string and the empty vector, respectively. If S is a set, then $x \xleftarrow{\$} S$ denotes the uniform selection of an element from S. If $\delta \in [0,1]$, then $b \xleftarrow{\delta} \{0,1\}$ denotes that b is assigned the outcome of a biased coin toss that returns 1 with probability δ and 0 with probability $1-\delta$. If A is a randomized algorithm, then $y \xleftarrow{\$} \mathsf{A}^{\mathsf{O}}(x)$ means that y is assigned the output of A on input x when given fresh coin tosses and access to oracle O.

SYNTAX. A *sequential aggregate signed data* (SASD) scheme is a tuple of three algorithms $\mathcal{SASD} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vf})$. Each signer generates his own key pair $(pk, sk) \xleftarrow{\$} \mathsf{Kg}(1^k)$ consisting of a public key pk and a secret key sk with security parameter $k \in \mathbb{N}$. The first signer in the sequence with key pair (pk_1, sk_1) creates the signed data Σ_1 for message M_1 by computing $\Sigma_1 \xleftarrow{\$} \mathsf{Sign}(sk_1, M_1)$. The n-th signer in the sequence receives from the $(n-1)$-st signer the aggregate signed data Σ_{n-1}, and adds his own signature on message M_n onto the aggregation by running $\Sigma_n \xleftarrow{\$} \mathsf{Sign}\big(sk_n, M_n, \Sigma_{n-1}\big)$. He then sends Σ_n on to the $(n+1)$-st signer. The verifier checks the validity of Σ_n by running the verification algorithm $(\vec{pk}, \vec{M}) \leftarrow \mathsf{Vf}(\Sigma_n)$. This algorithm either returns lists of n public keys $\vec{pk}$ and messages $\vec{M}$, indicating that the signature correctly authenticates message M_i under public key pk_i for $1 \leq i \leq n$, or returns $(\perp, \perp)$ to indicate rejection. Correctness requires that the $\mathsf{Vf}(\Sigma_n) = (\vec{pk}, \vec{M})$ with probability one for all messages $\vec{M}$ when all signers behave honestly as described above.

SECURITY. We take our inspiration for the security notion of SASD from the unforgeability notion of SAS schemes [LMRS04, BNN07]. The game begins with the generation of the key pair $(pk^*, sk^*) \xleftarrow{\$} \mathsf{Kg}(1^k)$ of the honest user that will be targeted in the attack. The forger F is given pk^* as input and has access to a signing oracle $\mathsf{Sign}(sk^*, \cdot, \cdot)$. This oracle, on input a message M_n and aggregate signed data Σ_{n-1}, returns $\Sigma_n \xleftarrow{\$} \mathsf{Sign}(sk^*, M_n, \Sigma_{n-1})$. In the random oracle model [BR93], the forger is additionally given oracle access to one or more random functions. At the end of its execution, F outputs its forgery Σ. The forger wins the game iff $\mathsf{Vf}(\Sigma) = (\vec{pk}, \vec{M}) \neq (\perp, \perp)$ and there exists an index $1 \leq i \leq |\vec{pk}|$ such that (1) $pk_i = pk^*$ and (2) F never made a signature query $\mathsf{Sign}(sk^*, M_i, \Sigma_{i-1})$ for any Σ_{i-1} such that $\mathsf{Vf}(\Sigma_{i-1}) = (\vec{pk}|_{i-1}, \vec{M}|_{i-1})$. We stress that our security notion does *not* use the KOSK assumption, i.e., the forger does *not* need to register the secret keys of corrupted signers involved in its signing queries or in its forgery.

The advantage of F is the probability that it wins the above game, where the probability is taken over the coins of Kg, Sign, and F itself. In the random oracle model, the probability is also over the choice of the random function(s) implemented by the random oracle(s). We say that F $(t, q_{\mathrm{S}}, n_{\max}, \epsilon)$-breaks $\mathcal{SASD}$ if it runs in time at most t, makes at most q_{S} signature queries, and has advantage at least ϵ, and aggregates contain at most $n_{\max}$ signatures. This means that the aggregate signed data that F submits to the signing oracle can contain at most $n_{\max}-1$ signatures, and that its forgery can contain at most $n_{\max}$ signatures. In the random oracle model, we additionally bound the number of queries that the adversary makes to each random oracle separately.

3 Our Main Construction

Claw-free permutations. A family of claw-free trapdoor permutations Π consists of a randomized permutation generation algorithm Pg that on input 1^k outputs tuples (π, ρ, π^{-1}) describing permutations π, ρ over domain $D_\pi = D_\rho$ of size $|D_\pi| \geq 2^{k-1}$, and the corresponding trapdoor information for the inverse permutation π^{-1}. There must exist efficient algorithms that given π, x compute $\pi(x)$ in time t_π, that given ρ, x compute $\rho(x)$, and that given π^{-1}, x compute $\pi^{-1}(x)$ for any $x \in D_\pi$. A claw-finding algorithm A is said to (t,ϵ)-break Π if it runs in time at most t and the probability

$$\Pr\left[\, \pi(x) = \rho(y) \mid (\pi, \rho, \pi^{-1}) \xleftarrow{\$} \mathsf{Pg}(1^k)\ ;\ (x,y) \xleftarrow{\$} \mathsf{A}(\pi,\rho) \,\right]$$

is at least ϵ.

Other ingredients. Let $k, \ell \in \mathbb{N}$ be security parameters, where ℓ is a system-wide parameter but k can be chosen by each signer independently as long as $k > \ell$. (Typical values for a security level of 80 bits in a factoring-based instantiation would be $k = 1024$ and $\ell = 160$.) Let Π be a family of claw-free trapdoor permutations so that associated to each permutation π in the family there exists an additive abelian group $\mathbb{G}_\pi \subseteq D_\pi$ such that $|\mathbb{G}_\pi| \geq 2^{k-1}$. Let $d = \min_{\pi \in \Pi}(|\mathbb{G}_\pi|/|D_\pi|)$ be the *minimal density* of $\mathbb{G}_\pi$ in D_π. We stress that π need *not* be a permutation over $\mathbb{G}_\pi$, and that π need *not* be homomorphic with respect to the group operation in $\mathbb{G}_\pi$. Let $\mathsf{enc}_\pi : \{0,1\}^* \to \{0,1\}^* \times \mathbb{G}_\pi$ an efficient encoding algorithm that breaks up a message M into a (shorter) message m and an element $\mu \in \mathbb{G}_\pi$, and let $\mathsf{dec}_\pi : \{0,1\}^* \times \mathbb{G}_\pi \to \{0,1\}^*$ be the corresponding decoding algorithm that reconstructs M from (m, μ). We require that the decoding function is injective, meaning that $\mathsf{dec}_\pi(m,\mu) = \mathsf{dec}_\pi(m',\mu') \Rightarrow (m,\mu) = (m',\mu')$. Finally, let $\mathrm{H} : \{0,1\}^* \to \{0,1\}^\ell$ and $\mathrm{G}_\pi : \{0,1\}^\ell \to \mathbb{G}_\pi$ be public hash functions modeled as random oracles.

Intuition. Before presenting our $\mathcal{SASD}$ scheme, we provide some intuition into the construction. First consider the following signature scheme with message recovery, that could be seen as a non-randomized generalization of PSS-R [BR96]. The signer's public key is a permutation π, the secret key is π^{-1}. To sign a message M, he computes $(m,\mu) \leftarrow \mathsf{enc}_\pi(M)$, $h \leftarrow \mathrm{H}(M)$, and $X \leftarrow \pi^{-1}(\mathrm{G}_\pi(h) + \mu)$. The signature consists of the pair $\sigma = (X, h)$. Given partial message m and signature σ, a verifier recomputes $\mu \leftarrow \pi(X) - \mathrm{G}_\pi(h)$, $M \leftarrow \mathsf{dec}_\pi(m,\mu)$, and returns M iff $\mathrm{H}(M) = h$. Observe that if the encoding is sufficiently dense ($d \approx 1$), then the net signing overhead is limited to $|h| = \ell$ bits, since the bandwidth of X is reused entirely for message recovery.

Two observations lead from this scheme to our $\mathcal{SASD}$ scheme. First, the type of data that can be "embedded" in X is not restricted to parts of the signed message; it could also be used for example to embed the signature of the previous signer. (The same idea actually underlies the $\mathcal{LMRS}$ scheme.) Second, suppose the signer wants to add a second signature on M_2 on top of $\sigma_1 = (X_1, h_1)$. One idea to keep the net overhead at a constant ℓ bits could be to use $h_2 \leftarrow h_1 \oplus \mathrm{H}(M_2)$ and let the overall signed data be $m_1\|m_2\|X_1\|X_2\|h_2$. The verifier can then recover M_2 from (m_2, X_2, h_2); h_1 from (h_2, M_2); and M_1 from (m_1, X_1, h_1). He accepts iff $\mathrm{H}(M_1) = h_1$. A number of additional tweaks would be needed to make this scheme secure (we do not make any claims about its security here), but this is the rough idea.

THE SCHEME. We associate to the above building blocks the $\mathcal{SAS}$ scheme as follows. Each signer generates a pair of claw-free trapdoor permutations $(\pi, \rho, \pi^{-1}) \xleftarrow{\$} \mathsf{Pg}(1^k)$. The public key is $pk \leftarrow \pi$, the secret signing key is $sk \leftarrow \pi^{-1}$. The aggregate signing and verification algorithms are given below.

Algorithm $\mathsf{Sign}^{\mathrm{H,G}}(\pi_n^{-1}, M_n, \Sigma_{n-1})$:
- If $n = 1$ then $\Sigma_0 \leftarrow (\vec{\varepsilon}, \varepsilon, \varepsilon, 0^\ell)$
- Parse Σ_{n-1} as $(\vec{\pi}, m_{n-1}, X_{n-1}, h_{n-1})$
- If $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_{n-1}) = (\perp, \perp)$ then return $\perp$
- $(m_n, \mu_n) \leftarrow \mathsf{enc}_{\pi_n}(M_n\|m_{n-1}\|X_{n-1})$
- $h_n \leftarrow h_{n-1} \oplus \mathrm{H}(\vec{\pi}|\pi_n\|M_n\|m_{n-1}\|X_{n-1})$
- $g_n \leftarrow \mathrm{G}_{\pi_n}(h_n)$
- $X_n \leftarrow \pi_n^{-1}(g_n + \mu_n)$
- Return $\Sigma_n \leftarrow (\vec{\pi}|\pi_n, m_n, X_n, h_n)$

Algorithm $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma)$:
- Parse Σ as $(\vec{\pi}, m_n, X_n, h_n)$ where $n = |\vec{\pi}|$
- For $i = n, \ldots, 1$ do
  - If $|\mathbb{G}_{\pi_i}| < 2^\ell$ then return $(\perp, \perp)$
  - $g_i \leftarrow \mathrm{G}_{\pi_i}(h_i)$; $\mu_i \leftarrow \pi_i(X_i) - g_i$
  - $M_i\|m_{i-1}\|X_{i-1} \leftarrow \mathsf{dec}_{\pi_i}(m_i, \mu_i)$
  - $h_{i-1} \leftarrow h_i \;\oplus\; \mathrm{H}(\vec{\pi}|_i\|M_i\|m_{i-1}\|X_{i-1})$
- If $(m_0, X_0, h_0) = (\varepsilon, \varepsilon, 0^\ell)$
- Then return $\big(\vec{\pi}, \vec{M} = (M_1, \ldots, M_n)\big)$
- Else return $(\perp, \perp)$.

EFFICIENCY. Note that the verification algorithm only contains a simple check on the output size of $\mathrm{G}_{\pi_i}(\cdot)$, but does not check whether $\mathbb{G}_{\pi_i} \subseteq D_{\pi_i}$ or whether π_i really describes a permutation over D_{π_i}. Indeed, unlike the $\mathcal{LMRS}$ scheme, our security analysis of the $\mathcal{SASD}$ scheme points out that an honest signer's security is *not* affected by adversarially generated keys of cosigners. This opens the way to much cheaper instantiations from uncertified permutations such as low-exponent RSA and factoring. The true reason for this difference only becomes clear in the details of the security proof, but we try to give some intuition here. The crucial difference between our $\mathcal{SASD}$ scheme and the $\mathcal{LMRS}$ scheme is that in $\mathcal{SASD}$ the data embedded in X_i, meaning (M_i, m_{i-1}, X_{i-1}), is passed as an argument to the hash function $\mathrm{H}(\cdot)$, as was done in the signature scheme with message recovery that we sketched before. This extra security measure cannot be applied to the $\mathcal{LMRS}$ scheme however because the embedded data (the previous signature) is only recovered *after* evaluating $\mathrm{H}(\cdot)$. Lysyanskaya et al. overcome this problem in the security proof by simulating random oracles mapping into $\mathbb{G}_\pi$ by choosing $x \xleftarrow{\$} D_\pi$ and returning $\pi(x)$. The correctness of this technique relies crucially on the fact that even adversarially generated π are permutations. The security proof of $\mathcal{SASD}$, on the other hand, only uses this simulation technique for the honestly generated permutation π^*. See the proof in Section 5 for details.

Also note that signers can independently choose their own value of the security parameter k. The system-wide parameter ℓ can be set to a comfortably high value like $\ell = 256$ or 512 without too much impact on performance. The exact overall bandwidth overhead depends on the length of the signed messages, the efficiency of the encoding algorithm, the family of permutations being used, the signers' security parameters $k_1, \ldots, k_n$ and the density d. For typical instantiations (see below), the net overhead varies from ℓ bits in case sufficiently long messages are being signed, meaning $|M_i| \geq k_i - k_{i-1}$, up to $\ell + \max(k_1, \ldots, k_n)$ bits for shorter messages. Finally, it is worth noting that the list of public keys $\vec{\pi}$ contained in Σ can of course be omitted from the transmitted data if the verifier already knows the public keys.

4 Instantiating our Construction

INSTANTIATIONS FROM RSA. An RSA key generator [RSA78] is a randomized algorithm $\mathsf{Kg}_{\mathrm{RSA}}$ that on input 1_{f}^k outputs tuples (N, e, d) where $N = pq$ is a k_{f}-bit product of two large primes and $ed = 1 \bmod \varphi(N)$. The RSA function $\pi(x) = x^e \bmod N$ is generally assumed to be a trapdoor one-way permutation over $D_\pi = \mathbb{Z}_N^*$, where d is the trapdoor that allows to compute $\pi^{-1}(x) = x^d \bmod N$. An algorithm A is said to (t, ϵ)-break the one-wayness of $\mathsf{Kg}_{\mathrm{RSA}}$ if it runs in time at most t and

$$\Pr\left[\, x^e = y \bmod N \mid (N, e, d) \stackrel{\$}{\leftarrow} \mathsf{Kg}_{\mathrm{RSA}} \,;\, y \stackrel{\$}{\leftarrow} \mathbb{Z}_N^* \,;\, x \stackrel{\$}{\leftarrow} \mathsf{A}(N, e, y) \,\right] \;\geq\; \epsilon \;.$$

One can associate to $\mathsf{Kg}_{\mathrm{RSA}}$ a claw-free permutation family Π by taking $\rho(x) = x^e \cdot y \bmod N$, where $y \stackrel{\$}{\leftarrow} \mathbb{Z}_N^*$. It is easy to see that if an algorithm A (t, ϵ)-breaks this claw-free permutation, then there exists an algorithm B that (t, ϵ)-breaks the one-wayness of $\mathsf{Kg}_{\mathrm{RSA}}$.

The most important advantage of our scheme over the $\mathcal{LMRS}$ scheme is that special RSA key generators can be used with small verification exponents, e.g. $e = 3$ or $e = 65537$. These have the advantage that the cost of a verification exponentiation (raising to exponent e) is reduced to that of a couple of multiplications.

Several options are available for the group $\mathbb{G}_\pi$, the additive group operation, the hash function $\mathrm{G}_\pi(\cdot)$, and the message encoding/decoding algorithms to be used. The most straightforward choice would be to use $\mathbb{G}_{N,e} = \mathbb{Z}_N^*$ with multiplication modulo N. A computationally more efficient choice however is to use $\mathbb{G}_{N,e} = \{0\|x \;:\; x \in \{0,1\}^{k_{\mathrm{f}}-1}\}$ with the XOR operation. To make optimal use of the bandwidth, one could also use $\mathbb{G}_{N,e} = \mathbb{Z}_N$ in combination with the addition modulo N. Alternatively, one can use the permutation family of [HOT04] to save one bit of bandwidth per signer, but this comes at the cost of doubling the verification and signing time.

To estimate efficiency (see overview in Table 1), we consider the instantiation with $e = 3$ and $\mathbb{G}_{N,e} = 0\|\{0,1\}^{k_{\mathrm{f}}-1}$ with the XOR operation, and we assume that signers use security parameter k_{f} or smaller. The exact bandwidth overhead depends on the amount of "recoverable" data μ_i, but varies between ℓ bits in the best case and $\ell + k_{\mathrm{f}} + n$ bits in the worst case. Typical bandwidth overhead will be around $\ell + n$ bits (for n signers using the same security parameter k_{f} and $|M_1| \geq k_{\mathrm{f}}$). The most expensive operations in signing and verification are the computations of π and π^{-1}, which for $e = 3$ take two modular multiplications and one modular exponentiation each, respectively. Ignoring cheap operations such as XORs and hash function evaluations, verifying aggregate signed data by n signers takes $2n$ modular multiplications. Placing a signature on top of an aggregate of n signers takes $2n$ multiplications (to verify the existing aggregate) and one exponentiation.

INSTANTIATIONS FROM FACTORING. Let $\mathsf{Kg}_{\mathrm{Wil}}$ be a randomized algorithm that on input 1^k outputs tuples (N, p, q) where $N = pq$ is a k-bit product of primes p, q such that $p = 3 \bmod 8$ and $q = 7 \bmod 8$. For such integers N, also called Williams integers, we have that the Legendre symbols $\left(\frac{-1}{p}\right) = \left(\frac{-1}{q}\right) = -1$, $\left(\frac{2}{p}\right) = -1$, and $\left(\frac{2}{q}\right) = +1$. Therefore -1 is a quadratic residue modulo N with Jacobi symbol $\left(\frac{-1}{N}\right) = +1$, and 2 is a non-quadratic residue with Jacobi symbol $\left(\frac{2}{N}\right) = -1$. Also, each quadratic residue modulo N has four square roots (x_1, x_2, x_3, x_4) of which $x_1 = -x_2 \bmod N$, $x_3 = -x_4 \bmod N$, $\left(\frac{x_1}{N}\right) = \left(\frac{x_2}{N}\right) = +1$, and $\left(\frac{x_2}{N}\right) = \left(\frac{x_3}{N}\right) = -1$. Consider the permutation $\pi : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined as

$$\pi(x) \;=\; \begin{cases} x^2 & \bmod N \quad \text{if } \left(\frac{x}{N}\right) = +1 \text{ and } x < N/2 \\ -x^2 & \bmod N \quad \text{if } \left(\frac{x}{N}\right) = +1 \text{ and } x > N/2 \\ 2x^2 & \bmod N \quad \text{if } \left(\frac{x}{N}\right) = -1 \text{ and } x < N/2 \\ -2x^2 & \bmod N \quad \text{if } \left(\frac{x}{N}\right) = -1 \text{ and } x > N/2 \;. \end{cases}$$

Note that the Jacobi symbol $\left(\frac{x}{N}\right)$ can be computed in time $O(|N|^2)$ without knowing the factorization of N, so the forward permutation is efficiently computable without knowing the trapdoor. The inverse permutation $\pi^{-1}(y)$ can be computed using trapdoor information p, q by finding $c \in \{1, -1, 2, -2\}$ such that y/c is a quadratic residue modulo N and computing the four square roots $(x_1, x_{-1}, x_2, x_{-2})$ of y/c modulo N, ordered such that $\left(\frac{x_1}{N}\right) = \left(\frac{x_{-1}}{N}\right) = +1$, $x_1 < x_{-1}$, $\left(\frac{x_2}{N}\right) = \left(\frac{x_{-2}}{N}\right) = -1$, $x_2 < x_{-2}$. The inverse of y is the root x_c. Since this is a permutation over $\mathbb{Z}_N^*$, the same group operations, hash functions and message encoding algorithms can be used as described for RSA above.

One can associate a family of claw-free trapdoor permutations to $\mathsf{Kg}_{\mathrm{Wil}}$ by taking $\rho(x) = \pi(x) \cdot r^2 \bmod N$ where $r \stackrel{\$}{\leftarrow} \mathbb{Z}_N^*$. Algorithm A is said to (t, ϵ)-factor $\mathsf{Kg}_{\mathrm{Wil}}$ if it runs in time at most t and

$$\Pr\Big[\, x \in \{p, q\} \mid (N, p, q) \stackrel{\$}{\leftarrow} \mathsf{Kg}_{\mathrm{Wil}} \,;\, x \stackrel{\$}{\leftarrow} \mathsf{A}(N) \,\Big] \;\geq\; \epsilon \;.$$

Given a claw $\pi(a) = \rho(b)$, one can see that $a/b \bmod N$ is a square root of r^2, which with probability $1/2$ is different from $\pm r \bmod N$ and thereby reveals the factorization of N. Therefore, if an algorithm A (t, ϵ)-breaks the claw-free permutation, then there exists an algorithm B that $(t, \epsilon/2)$-factors $\mathsf{Kg}_{\mathrm{Wil}}$.

5 Security of our Construction

We prove the security of the $\mathcal{SASD}$ scheme in the random oracle model under the claw-freeness of the permutation family Π. The following theorem gives a formal security statement with concrete security bounds.

Theorem 5.1 If there exists a forger F that $(t, q_{\mathrm{S}}, q_{\mathrm{H}}, q_{\mathrm{G}}, n_{\max}, \epsilon)$-breaks $\mathcal{SASD}$ in the random oracle model, then there exists a claw-finding algorithm A that (t', ϵ')-breaks Π with

$$\begin{aligned} \epsilon' &\;\geq\; \frac{\epsilon}{e(q_{\mathrm{S}}+1)} - \frac{4\big(q_{\mathrm{H}} + q_{\mathrm{G}} + 2n_{\max}(q_{\mathrm{S}}+1)\big)^2}{2^\ell} \\ t' &\;\leq\; t + (1/d + 2)\big(q_{\mathrm{H}} + 2n_{\max}(q_{\mathrm{S}}+1) + n_{\max}\big) \cdot t_\pi \;. \end{aligned}$$

We prove Theorem 5.1 in two steps. First, we restrict our attention to a particular class of forgers that we call *sequential* forgers, defined in Definition 5.2. In Lemma 5.3 we show that for any (non-sequential) forger F there exists a sequential forger S with about the same success probability and running time. Next, we show in Lemma 5.5 how a sequential forger can be used to find a claw in Π. The theorem then follows directly by combining Lemma 5.3 and Lemma 5.5.

Definition 5.2 We say that a forger S against $\mathcal{SASD}$ is *sequential* if:

1. it never makes the same $\mathrm{H}(\cdot)$, $\mathrm{G}.(\cdot)$, or $\mathsf{Sign}(\cdot,\cdot)$ query twice, where $\mathrm{H}(\cdot)$ denotes a query to the random oracle for H, and where $\mathrm{G}.(\cdot)$ denotes a query to the random oracle for G_π for some permutation π;

2. it only makes $\mathrm{H}(\cdot)$ queries of the form $\mathrm{H}(\vec{\pi}\|M_n\|m_{n-1}\|X_{n-1})$ such that $n = |\vec{\pi}| \leq n_{\max}$ and $|\mathbb{G}_{\pi_i}| \geq 2^{\ell}$ for all $1 \leq i \leq n$;

3. for each query $\mathrm{H}(Q_n) = \mathrm{H}(\vec{\pi}\|M_n\|m_{n-1}\|X_{n-1})$ there exists a unique sequence of queries $Q_1 = (\vec{\pi}|_1\|M_1\|\varepsilon\|\varepsilon), Q_2 = (\vec{\pi}|_2\|M_2\|m_1\|X_1), \ldots, Q_{n-1} = (\vec{\pi}|_{n-1}\|M_{n-1}\|m_{n-2}\|X_{n-2})$ such that S previously made queries $\mathrm{H}(Q_1), \ldots, \mathrm{H}(Q_{n-1})$, in that order, such that

$$\mathsf{dec}_{\pi_i}\big(m_i\ ,\ \pi_i(X_i) - \mathrm{G}_{\pi_i}(h_i)\big) \ = \ M_i\|m_{i-1}\|X_{i-1}\ ,$$

where $h_i = h_{i-1} \oplus \mathrm{H}(Q_i)$ for $1 \leq i \leq n$ and such that $(m_0, X_0, h_0) = (\varepsilon, \varepsilon, 0^{\ell})$.

4. it only makes signing queries $\mathsf{Sign}(\pi^{-1}, M_n, \Sigma_{n-1})$ for valid signed data $\Sigma_{n-1} = (\vec{\pi}, m_{n-1}, X_{n-1}, h_{n-1})$, meaning that $n = |\vec{\pi}| + 1 \leq n_{\max}$ and $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_{n-1}) \neq (\perp, \perp)$. Also, before making such a signing query, it makes a random oracle query $\mathrm{H}(\vec{\pi}\|M_{n-1}\|m_{n-2}\|X_{n-2})$ and all random oracle queries needed for the verification $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_{n-1})$;

5. it only outputs valid forgeries $\Sigma = (\vec{\pi}, m_n, X_n, h_n)$, meaning that $n = |\vec{\pi}| \leq n_{\max}$, that $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma) = (\vec{\pi}, \vec{M}) \neq (\perp, \perp)$, and that there exists $1 \leq i \leq n$ such that $\pi_i = \pi^*$ and S never made a signing query $\mathsf{Sign}(\pi^{*-1}, M_i, \Sigma_{i-1})$ for any Σ_{i-1} such that $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_{i-1}) = (\vec{\pi}|_{i-1}, \vec{M}|_{i-1})$. Also, before halting, it makes all random oracle queries needed for the verification $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma)$.

The following lemma shows that for any non-sequential forger F, there exists a sequential forger S with approximately the same success probability and running time as F.

Lemma 5.3 If there exists a forger F that $(t, q_{\mathrm{S}}, q_{\mathrm{H}}, q_{\mathrm{G}}, n_{\max}, \epsilon)$-breaks $\mathcal{SASD}$ in the random oracle model, then there exists a sequential forger S that $(t', q_{\mathrm{S}}, q'_{\mathrm{H}}, q'_{\mathrm{G}}, n_{\max}, \epsilon')$-breaks $\mathcal{SASD}$ in the random oracle model with

$$\begin{aligned} \epsilon' &\geq \epsilon - \frac{2\big(q_{\mathrm{H}} + q_{\mathrm{G}} + 2n_{\max}(q_{\mathrm{S}}+1)\big)^2}{2^{\ell}} && (1)\\ q'_{\mathrm{H}} &\leq q_{\mathrm{H}} + n_{\max}(q_{\mathrm{S}}+1) \\ q'_{\mathrm{G}} &\leq q_{\mathrm{H}} + q_{\mathrm{G}} + 2n_{\max}(q_{\mathrm{S}}+1) \\ t' &\leq t + (q_{\mathrm{H}} + 2n_{\max}(q_{\mathrm{S}}+1)) \cdot t_{\pi}\ . \end{aligned}$$

Proof: We first give an informal description of the sequential forger S to provide some intuition into its strategy, and then give a formal description to derive the exact concrete security bounds. Given a non-sequential forger F, we build a sequential forger S as follows. Algorithm S is given input π^* and access to oracles $\mathrm{H}'(\cdot)$, $\mathrm{G}'_{\cdot}(\cdot)$, and $\mathsf{Sign}'(\pi^{*-1}, \cdot, \cdot)$. It runs F on the same input π^* and simulates responses to F's $\mathrm{H}(\cdot)$, $\mathrm{G}_{\cdot}(\cdot)$ and $\mathsf{Sign}(\pi^{*-1}, \cdot, \cdot)$ oracle queries.

To satisfy Property 1 of Definition 5.2, S stores all previous responses to F's oracle queries in associative tables, retrieving the appropriate response from these tables when F asks the same query again. Note that the Sign algorithm is deterministic, so in a real attack repeating the same query to the signing oracle would result in the same signature being returned as well. Property 2 is satisfied by returning random values for F's malformed $\mathrm{H}(\cdot)$ queries. To answer F's $\mathrm{G}_{\cdot}(\cdot)$ queries, S simply relays responses from its own $\mathrm{G}'_{\cdot}(\cdot)$ oracle.

Correctly formed $\mathrm{H}(\cdot)$ queries are treated in a more complicated manner. S maintains a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as illustrated in Figure 1 . Each node is uniquely identified by a tuple

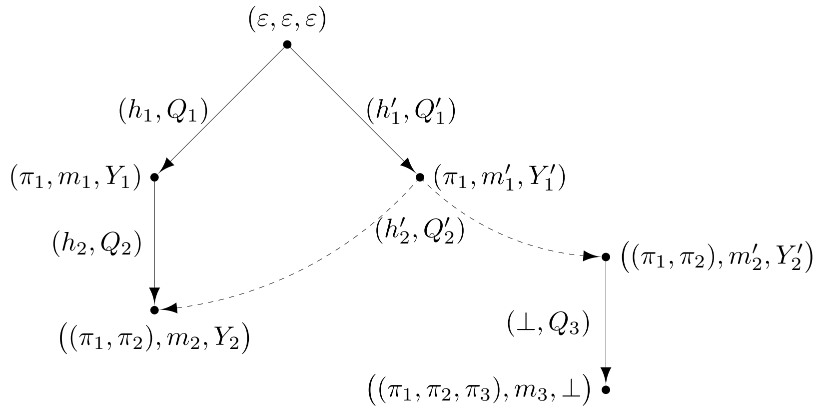

Figure 1: The graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ maintained by algorithm S. The solid edges indicate the state of $\mathcal{G}$ after F made sequential $\mathrm{H}(\cdot)$ queries $Q_1 = \pi_1 \| M_1 \| \varepsilon \| \varepsilon$, $Q_2 = (\pi_1, \pi_2) \| M_2 \| m_1 \| X_1$, $Q'_1 = \pi_1 \| M'_1 \| \varepsilon \| \varepsilon$, and a non-sequential query $Q_3 = (\pi_1, \pi_2, \pi_3) \| M_3 \| m_2 \| X_2$. The dashed edges depict the problematic cases when at that point F makes a new query $\mathrm{H}(Q'_2) = \mathrm{H}\big((\pi_1, \pi_2) \| M'_2 \| m'_1 \| X'_1\big)$ that causes event $\textsc{Bad}$ to occur.

$(\vec{\pi}, m, Y) \in \mathcal{V}$, and each edge is uniquely identified by a tuple $(h, Q) \in \mathcal{E}$. We explicitly allow multiple directed edges between the same pair of nodes. Initially, the graph only contains a so-called root node $(\varepsilon, \varepsilon, \varepsilon)$. The idea is that all queries $\mathrm{H}(Q_n)$ satisfying Property 3, so-called *sequential* queries, appear in edges in a tree rooted at $(\varepsilon, \varepsilon, \varepsilon)$, while all non-sequential queries appear in edges not connected to $(\varepsilon, \varepsilon, \varepsilon)$. We refer to the tree rooted at $(\varepsilon, \varepsilon, \varepsilon)$ as the *sequential tree*. Property 3 is then enforced by letting S return $\mathrm{H}'(Q_n)$ for queries in the sequential tree, and random values for all other queries.

When F makes a new query $\mathrm{H}(Q_n) = \mathrm{H}(\vec{\pi} \| M_n \| m_{n-1} \| X_{n-1})$, S updates the graph $\mathcal{G}$ as follows. If $n = 1$ and $m_0 = X_0 = \varepsilon$, then the query trivially satisfies Property 3, so it creates a new edge (h_1, Q_1) with the root as tail node and a new node $v_{\mathrm{h}} = (\pi_1, m_1, Y_1)$ as head node, where $(m_1, \mu_1) = \mathsf{enc}_{\pi_1}(M_1 \| \varepsilon \| \varepsilon)$, $h_1 = \mathrm{H}(Q_1)$, and $Y_1 = \mu_1 + \mathrm{G}_{\pi_1}(h_1)$. When $1 < n \leq n_{\max}$, algorithm S searches the graph for a node $v_{\mathrm{t}} = (\vec{\pi}|_{n-1}, m_{n-1}, Y_{n-1})$ where $Y_{n-1} = \pi_{n-1}(X_{n-1})$. If such a node exists and it is in the sequential tree, then let (h_{n-1}, Q_{n-1}) be the incoming edge into v_{t}. We have that $\pi_{n-1}(X_{n-1}) = Y_{n-1} = \mu_{n-1} + \mathrm{G}_{\pi_{n-1}}(h_{n-1})$, so the requirements of Property 3 are satisfied by the sequence of queries $(Q_1, \ldots, Q_{n-1})$ on the path from the root to v_{t}. Algorithm S creates a new edge (h_n, Q_n) with tail node v_{t} and head node $v_{\mathrm{h}} = (\vec{\pi}, m_n, Y_n)$ where $h_n = h_{n-1} \oplus \mathrm{H}(Q_n)$, $(m_n, \mu_n) = \mathsf{enc}_{\pi_n}(M_n \| m_{n-1} \| X_{n-1})$, and $Y_n = \mu_n + \mathrm{G}_{\pi_n}(h_n)$. If v_{t} is not in the sequential tree, or if no such node v_{t} exists in the graph, then the query is deemed non-sequential. Algorithm S returns a random value as the random oracle response, and adds a new edge $(\perp, Q_n)$ to the graph with tail node $v_{\mathrm{t}} = (\vec{\pi}|_{n-1}, m_{n-1}, Y_{n-1})$ where $Y_n = \pi_{n-1}(X_{n-1})$ and head node $v_{\mathrm{h}} = (\vec{\pi}, m_n, \perp)$ where $(m_n, \mu_n) = \mathsf{enc}_{\pi_n}(M_n \| m_{n-1} \| X_{n-1})$, adding these nodes to the graph if they did not yet exist.

The creation of a new edge, however, should not violate the invariants that only sequential queries are represented by edges in the sequential tree, and that all of these queries were responded to using outputs of $\mathrm{H}'(\cdot)$. Two types of problems that can occur are illustrated by the dashed arrows for query Q'_2 in Figure 1 . The left arrow illustrates the situation when Q'_2 is such that the head node v_{h} of the new edge coincides with an existing node in the sequential tree. This is a problem, because if F later makes a query $\mathrm{H}(Q_3')$ that "connects" to v_{h}, then there exist two different sequences (Q_1, Q_2) and (Q_1', Q_2') that satisfy the requirements of Property 3, violating the uniqueness requirement. The right dashed arrow in Figure 1 illustrates the situation when v_{h} coincides with an existing node that is not part of the sequential tree. The newly created edge would suddenly incorporate v_{h} into the sequential tree, but this violates the invariant because S responded the query $\mathrm{H}(Q_3)$ with a random value, rather than with an output of $\mathrm{H}'(\cdot)$.

To preempt these problems, S aborts its execution whenever a new edge is added to the sequential tree with a head node that already exists in $\mathcal{G}$. We say that event $\textsc{Bad}$ occurs when this happens.

Claim 5.4 If Σ_n are valid signed data, meaning $n \leq n_{\max}$ and $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_n) \neq (\perp, \perp)$, and event $\textsc{Bad}$ does not occur, then all random oracle queries involved in the evaluation of $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_n)$ are sequential.

Proof: Let Σ_n be parsed as $(\vec{\pi}, m_n, X_n, h_n)$. We prove the claim by induction on the number of signatures n contained in Σ_n. The claim clearly holds for $n = 1$, because in this case the verification involves only a single query $\mathrm{H}(\pi_1 \| M_1 \| \varepsilon \| \varepsilon)$ that is always sequential.

Suppose the claim is true for all signed data containing up to $n-1$ signatures. Let $Q_n, \ldots, Q_1$ be the $\mathrm{H}(\cdot)$ queries made when evaluating $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_n) = (\vec{\pi}, \vec{M})$, where $Q_i = \vec{\pi}|_i \| M_i \| m_{i-1} \| X_{i-1}$, and let $h_1', \ldots, h_{n-1}'$ be the intermediate values obtained during the evaluation. If Σ_n is valid, then $\Sigma_{n-1} = (\vec{\pi}|_{n-1}, m_{n-1}, X_{n-1}, h_{n-1}')$ is also valid, namely $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_{n-1}) = (\vec{\pi}|_{n-1}, \vec{M}|_{n-1})$. By the induction hypothesis, F must thus have made the queries $\mathrm{H}(Q_1), \ldots, \mathrm{H}(Q_{n-1})$ sequentially, so they are represented in the graph $\mathcal{G}$ by edges $(h_1, Q_1), \ldots, (h_{n-1}, Q_{n-1})$ in the sequential tree. Clearly, we have that $h_i = h_i' = \mathrm{H}(Q_1) \oplus \ldots \oplus \mathrm{H}(Q_i)$ for $1 \leq i \leq n-1$, and that $h_n = h_{n-1} \oplus \mathrm{H}(Q_n)$.

Suppose for contradiction that F queries $\mathrm{H}(Q_n)$ non-sequentially, so it queries $\mathrm{H}(Q_n)$ at least before it queries $\mathrm{H}(Q_{n-1})$. At the moment that F queried $\mathrm{H}(Q_n)$, S created an edge $(\perp, Q_n)$ with tail node $\big(\vec{\pi}|_{n-1}, m_{n-1}, Y_{n-1} = \pi_{n-1}(X_{n-1})\big)$ not connected to the sequential tree. When F subsequently queries $\mathrm{H}(Q_{n-1})$, S adds an edge (h_{n-1}, Q_{n-1}) to the sequential tree with head node $\big(\vec{\pi}|_{n-1}, m_{n-1}, Y_{n-1}' = \mu_{n-1} + \mathrm{G}_{\pi_{n-1}}(h_{n-1})\big)$. Since $\mathsf{dec}_{\pi_{n-1}}$ is injective however, there is only a single value of μ_{n-1} so that $\mathsf{dec}_{\pi_{n-1}}(m_{n-1}, \mu_{n-1}) = M_{n-1} \| m_{n-2} \| X_{n-2}$. In the verification of Σ_n, this value is recovered as $\mu_{n-1} = \pi_{n-1}(X_{n-1}) - \mathrm{G}_{\pi_{n-1}}(h_{n-1})$, so we have that $\mu_{n-1} = \pi_{n-1}(X_{n-1}) - \mathrm{G}_{\pi_{n-1}}(h_{n-1}) = Y_{n-1}' - \mathrm{G}_{\pi_{n-1}}(h_{n-1})$ and hence, because $\mathbb{G}_{\pi_{n-1}}$ is a group, that $Y_{n-1} = Y_{n-1}'$. This however means that the head node of (h_{n-1}, Q_{n-1}) coincides with the tail node of (h_n, Q_n), causing event $\textsc{Bad}$ to occur. So if $\textsc{Bad}$ does not occur, we conclude that F must query $\mathrm{H}(Q_n)$ after $\mathrm{H}(Q_{n-1})$, meaning sequentially. ■

When F makes a signing query $\mathsf{Sign}(\pi^{*-1}, M_n, \Sigma_{n-1})$, algorithm S enforces Property 4 of Definition 5.2 by first verifying Σ_{n-1} and simulating an additional query $\mathrm{H}(\vec{\pi}|\pi^* \| M_n \| m_{n-1} \| X_{n-1})$. Only if Σ_{n-1} verifies correctly does S consult its own signing oracle; it relays the response of $\mathsf{Sign}'(\pi^{*-1}, M_n, \Sigma_{n-1})$ to F. Note that by Claim 5.4, if the event $\textsc{Bad}$ does not occur, all $\mathrm{H}(\cdot)$ queries involved in the verification $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_{n-1}) = (\vec{\pi}, \vec{M}) \neq (\perp, \perp)$ are sequential, so we also have that $\mathsf{Vf}^{\mathrm{H',G'}}(\Sigma_{n-1}) = (\vec{\pi}, \vec{M})$.

Finally, S ensures Property 5 by first verifying the forgery, simulating the necessary random oracle queries, and checking that the conditions with respect to the previous $\mathsf{Sign}(\pi^{*-1}, \cdot, \cdot)$

Algorithm $\mathsf{S}^{\mathrm{H}',\mathrm{G}',\mathsf{Sign}'}(\pi^*)$:

001 $\mathcal{V} \leftarrow \{(\varepsilon,\varepsilon,\varepsilon)\}$; $\mathcal{E} \leftarrow \emptyset$; $\mathcal{G} \leftarrow (\mathcal{V},\mathcal{E})$
002 $\Sigma \xleftarrow{\$} \mathsf{F}^{\mathrm{H},\mathrm{G},\mathsf{Sign}}(\pi^*)$, answering queries
003 $\quad$ $\mathrm{H}(\cdot)$ using $\mathsf{H\text{-}Sim}(\cdot)$
004 $\quad$ $\mathrm{G}_{\cdot}(\cdot)$ using $\mathsf{G\text{-}Sim}(\cdot,\cdot)$
005 $\quad$ $\mathsf{Sign}(\pi^{*-1},\cdot,\cdot)$ using $\mathsf{S\text{-}Sim}(\cdot,\cdot)$
006 $(\vec{\pi},\vec{M}) \leftarrow \mathsf{Vf}^{\mathsf{H\text{-}Sim},\mathsf{G\text{-}Sim}}(\Sigma)$
007 If $(\vec{\pi},\vec{M}) = (\bot,\bot)$ or $\nexists\, 1 \le i \le |\vec{\pi}|$: $\big(\pi_i \neq \pi^*$ and $\nexists\, \Sigma_{i-1} : (ST[M_i,\Sigma_{i-1}] \neq \bot$ and $VT[\Sigma_{i-1}] = (\vec{\pi}|_{i-1},\vec{M}|_{i-1}))\,\big)$
008 Then abort
009 Output Σ

Subroutine $\mathsf{S\text{-}Sim}(M_n,\Sigma_{n-1})$:

101 If Σ_{n-1} not parsable as $(\vec{\pi},m_{n-1},X_{n-1},h_{n-1})$ where $n = |\vec{\pi}|+1$ or $n > n_{\max}$ then return $\bot$
102 If $ST[M_n,\Sigma_{n-1}] = \bot$ then
103 $\quad$ $\mathsf{H\text{-}Sim}(\vec{\pi}|\pi^*\|M_n\|m_{n-1}\|X_{n-1})$
104 $\quad$ $VT[\Sigma_{n-1}] \leftarrow \mathsf{Vf}^{\mathsf{H\text{-}Sim},\mathsf{G\text{-}Sim}}(\Sigma_{n-1})$
105 $\quad$ If $VT[\Sigma_{n-1}] = (\bot,\bot)$ then return $\bot$
106 $\quad$ $ST[M_n,\Sigma_{n-1}] \leftarrow \mathsf{Sign}'(\pi^{*-1},M_n,\Sigma_{n-1})$
107 Return $ST[M_n,\Sigma_{n-1}]$

Subroutine $\mathsf{H\text{-}Sim}(Q_n)$:

201 If $HT[Q_n] \neq \bot$ then return $HT[Q_n]$
202 If Q_n not parsable as $\vec{\pi}\|M_n\|m_{n-1}\|X_{n-1}$ where $n = |\vec{\pi}|$ or $n > n_{\max}$ or $|\mathbb{G}_{\pi_n}| < 2^\ell$
203 Then $HT[Q_n] \xleftarrow{\$} \{0,1\}^\ell$
204 Else
205 $\quad$ $(m_n,\mu_n) \leftarrow \mathsf{enc}_{\pi_n}(M_n\|m_{n-1}\|X_{n-1})$
206 $\quad$ If $n=1$ then $h_0 \leftarrow 0^\ell$; $v_{\mathrm{t}} \leftarrow (\varepsilon,\varepsilon,\varepsilon)$
207 $\quad$ Else
208 $\qquad$ $v_{\mathrm{t}} \leftarrow (\vec{\pi}|_{n-1},m_{n-1},\pi_{n-1}(X_{n-1}))$
209 $\qquad$ $h_{n-1} \leftarrow \mathsf{HeadOf}(v_{\mathrm{t}})$
210 $\quad$ If $h_{n-1} \neq \bot$ then // *sequential query*
211 $\qquad$ $HT[Q_n] \leftarrow \mathrm{H}'(Q_n)$
212 $\qquad$ $h_n \leftarrow h_{n-1} \oplus HT[Q_n]$
213 $\qquad$ $g_n \leftarrow \mathsf{G\text{-}Sim}(\pi_n,h_n)$
214 $\qquad$ $v_{\mathrm{h}} \leftarrow (\vec{\pi},m_n,\mu_n+g_n)$
215 $\qquad$ If $v_{\mathrm{h}} \in \mathcal{V}$ then abort // Bad *occurred*
216 $\quad$ Else // *non-sequential query*
217 $\qquad$ $HT[Q_n] \xleftarrow{\$} \{0,1\}^\ell$
218 $\qquad$ $v_{\mathrm{h}} \leftarrow (\vec{\pi},m_n,\bot)$; $h_n \leftarrow \bot$
219 $\quad$ $\mathsf{NewEdge}(v_{\mathrm{t}},v_{\mathrm{h}},(h_n,Q_n))$
220 Return $HT[Q_n]$

Subroutine $\mathsf{G\text{-}Sim}(\pi,h)$:

301 If $GT[\pi,h] = \bot$ then $GT[\pi,h] \leftarrow \mathrm{G}'_\pi(h)$
302 Return $GT[\pi,h]$

Figure 2: Pseudo-code description of the strictly sequential forger S of Lemma 5.3.

---

queries hold. Again, we rely here on Claim 5.4 to guarantee that if event Bad does not occur, then any valid signed data under random oracles $\mathrm{H}(\cdot), \mathrm{G}_{\cdot}(\cdot)$ is also valid under $\mathrm{H}'(\cdot), \mathrm{G}'_{\cdot}(\cdot)$.

It is clear that S is successful in breaking $\mathcal{SASD}$ whenever F is and event Bad does not occur. To prove the concrete bounds stated in Lemma 5.3 and to correctly analyze the probability that event Bad occurs, we give a more formal pseudocode description of the sequential forger S that we just described in Figure 2 .

Algorithm S runs F on input π^* and uses subroutines $\mathsf{H\text{-}Sim}$, $\mathsf{G\text{-}Sim}$, and $\mathsf{S\text{-}Sim}$ to simulate responses to the $\mathrm{H}(\cdot)$, $\mathrm{G}_{\cdot}(\cdot)$, and $\mathsf{Sign}(\cdot)$ queries made by F, respectively. We assume the availability of two subroutines $\mathsf{NewEdge}$ and HeadOf to manipulate the graph $\mathcal{G}$. The subroutine $\mathsf{NewEdge}(v_1,v_2,e)$ creates a new edge e from node v_1 to node v_2, creating these nodes if they do not yet exist. The subroutine $\mathsf{HeadOf}(v)$ checks whether there exists an edge $e=(h,X) \in \mathcal{E}$ that has v as a head node. If so, then HeadOf returns the value h; if not, it returns $\bot$.

We first show the claimed bounds on S's resource usage. For each signing query of F, S makes at most one query to its own signing oracle, so

$$q'_{\mathrm{S}} \;\le\; q_{\mathrm{S}} \;.$$

Algorithm S makes at most $q_{\mathrm{H}} + n_{\max}q_{\mathrm{S}} + n_{\max}$ calls to $\mathsf{H\text{-}Sim}$ (see lines 003, 006, 103, and 104), and at most $q_{\mathrm{H}} + q_{\mathrm{G}} + (2n_{\max}-1)q_{\mathrm{S}} + 2n_{\max}$ calls to $\mathsf{G\text{-}Sim}$ (see lines 004, 006, 104, and 213).

For each call to $\mathsf{H\text{-}Sim}$ and $\mathsf{G\text{-}Sim}$, S makes at most one query to $\mathrm{H}'(\cdot)$ and $\mathrm{G}'_{\cdot}(\cdot)$, respectively, so we have that

$$\begin{aligned} q'_{\mathrm{H}} &\leq q_{\mathrm{H}} + n_{\max}(q_{\mathrm{S}}+1) \\ q'_{\mathrm{G}} &\leq q_{\mathrm{H}} + q_{\mathrm{G}} + 2n_{\max}(q_{\mathrm{S}}+1) \ . \end{aligned}$$

For the running time we ignore all costs other than permutation evaluations, which take time t_π each. Algorithm S evaluates up to $n_{\max}$ permutations on line 006; up to $(n_{\max}-1)q_{\mathrm{S}}$ on line 104; and up to $q_{\mathrm{H}} + n_{\max}(q_{\mathrm{S}}+1)$ on line 208; summing up to

$$t' \leq t + (q_{\mathrm{H}} + 2n_{\max}(q_{\mathrm{S}}+1)) \cdot t_\pi \ .$$

For the advantage ϵ' of S in breaking $\mathcal{SASD}$ we have that

$$\begin{aligned} \epsilon' &= \Pr[\,\mathsf{S} \text{ wins}\,] \\ &= \Pr\left[\,\mathsf{S} \text{ wins} \mid \overline{\textsc{Bad}}\,\right] \cdot \Pr\left[\,\overline{\textsc{Bad}}\,\right] + \Pr[\,\mathsf{S} \text{ wins} \wedge \textsc{Bad}\,] \\ &\geq \Pr\left[\,\mathsf{S} \text{ wins} \mid \overline{\textsc{Bad}}\,\right] \cdot (1 - \Pr[\,\textsc{Bad}\,]) \\ &\geq \Pr\left[\,\mathsf{S} \text{ wins} \mid \overline{\textsc{Bad}}\,\right] - \Pr[\,\textsc{Bad}\,] \end{aligned} \tag{2}$$

From the strategy of S and Claim 5.4 it is clear that in the event $\overline{\textsc{Bad}}$ the simulation of F's environment provided by S is perfect, and that S wins whenever F wins:

$$\Pr\left[\,\mathsf{S} \text{ wins} \mid \overline{\textsc{Bad}}\,\right] \geq \Pr[\,\mathsf{F} \text{ wins}\,] = \epsilon \ . \tag{3}$$

Equation (1) easily follows from Equation (2), Equation (3) and the following bound on the probability that event $\textsc{Bad}$ occurs:

$$\Pr[\,\textsc{Bad}\,] \leq \frac{2\big(q_{\mathrm{H}} + q_{\mathrm{G}} + 2n_{\max}(q_{\mathrm{S}}+1)\big)^2}{2^\ell} \ .$$

To see why this bound is correct, observe that when event $\textsc{Bad}$ occurs on line 215, $HT[Q_n]$ was just assigned a fresh random value on line 211, so $h_n = h_{n-1} \oplus HT[Q_n]$ is random as well. We distinguish between the case that the call to $\mathsf{G\text{-}Sim}$ on line 213 resulted in a new entry being added to $GT[\cdot,\cdot]$, and the case that it didn't. In the latter case, h_n must already have appeared in $GT[\cdot,\cdot]$, but this happens with probability at most $|GT|/2^\ell \leq (q_{\mathrm{H}} + q_{\mathrm{G}} + 2n_{\max}(q_{\mathrm{S}}+1))/2^\ell$. In the former case, g_n is a random value from $\mathbb{G}_{\pi_n}$, so the probability that $Y_n = \mu_n + g_n$ already occurred as part of a node identifier is at most $|\mathcal{V}|/|\mathbb{G}_{\pi_n}| \leq (q_{\mathrm{H}} + n_{\max}(q_{\mathrm{S}}+1))/2^\ell$. Summing up over all $q_{\mathrm{H}} + n_{\max}(q_{\mathrm{S}}+1)$ calls to $\mathsf{H\text{-}Sim}$ gives

$$\begin{aligned} \Pr[\,\textsc{Bad}\,] &\leq (q_{\mathrm{H}} + n_{\max}(q_{\mathrm{S}}+1)) \cdot \left(\frac{q_{\mathrm{H}} + q_{\mathrm{G}} + 2n_{\max}(q_{\mathrm{S}}+1)}{2^\ell} + \frac{q_{\mathrm{H}} + n_{\max}(q_{\mathrm{S}}+1)}{2^\ell}\right) \\ &\leq \frac{2\big(q_{\mathrm{H}} + q_{\mathrm{G}} + 2n_{\max}(q_{\mathrm{S}}+1)\big)^2}{2^\ell} \ . \end{aligned}$$

■

Lemma 5.3 says that we can safely restrict our attention to sequential forgers. The next lemma shows that any such forger can be turned into a claw-finding algorithm for Π. The proof is given below, and reuses ideas from [BR96, Cor00, LMRS04, BNN07].

Lemma 5.5 If there exists a sequential forger S that $(t, q_{\mathrm{S}}, q_{\mathrm{H}}, q_{\mathrm{G}}, n_{\max}, \epsilon)$-breaks $\mathcal{SASD}$, then there exists a claw-finding algorithm A that (t', ϵ')-breaks Π for

$$\begin{aligned} \epsilon' &\geq \frac{\epsilon}{e(q_{\mathrm{S}}+1)} - \frac{q_{\mathrm{H}}(q_{\mathrm{H}}+q_{\mathrm{G}})}{2^{\ell}} \\ t' &\leq t + ((1/d+1)q_{\mathrm{H}} + n_{\max}) \cdot t_{\pi} \ . \end{aligned}$$

Proof: Given a sequential forger S against $\mathcal{SASD}$, consider the following claw-finding algorithm A against Π. Algorithm A maintains initially empty associative arrays $HT[\cdot]$ and $GT[\cdot,\cdot]$. On input π^*, ρ^*, algorithm A runs S on target public key π^*, and responds to its oracle queries as follows:

Random oracle query $\mathrm{H}(Q_n)$**:** Parse Q_n as $\vec{\pi}\|M_n\|m_{n-1}\|X_{n-1}$. If $n > 1$, then A finds the unique sequence of queries $(Q_1, \ldots, Q_{n-1})$ as per Property 3 of a sequential forger, and looks up $HT[Q_{n-1}] = (c, x, h_{n-1})$. If $n = 1$, it sets $h_0 \leftarrow 0^{\ell}$.

If $\pi_n \neq \pi^*$ then A chooses $h \stackrel{\$}{\leftarrow} \{0,1\}^{\ell}$, computes $h_n \leftarrow h \oplus h_{n-1}$, stores $HT[Q_n] \leftarrow (\perp, \perp, h_n)$, and returns h to S.

If $\pi_n = \pi^*$ then A chooses $h \stackrel{\$}{\leftarrow} \{0,1\}^{\ell}$ and $c \stackrel{\delta}{\leftarrow} \{0,1\}$, and computes $(m_n, \mu_n) = \mathsf{enc}_{\pi^*}(M_n\|m_{n-1}\|X_{n-1})$ and $h_n \leftarrow h \oplus h_{n-1}$. If $c = 0$ then A repeatedly chooses $x \stackrel{\$}{\leftarrow} D_{\pi^*}$ and computes $g_n \leftarrow \pi^*(x) - \mu_n$ until $g_n \in \mathbb{G}_{\pi^*}$. If $c = 1$ then A repeatedly chooses $x \stackrel{\$}{\leftarrow} D_{\pi^*}$ and computes $g_n \leftarrow \rho^*(x) - \mu_n$ until $g_n \in \mathbb{G}_{\pi^*}$. (Each of these loops will require $1/d$ iterations on average.) If $GT[\pi^*, h_n]$ is already defined, then we say that event $\textsc{Bad}_1$ occurred and A aborts; otherwise, it sets $GT[\pi^*, h_n] \leftarrow g_n$. It stores $HT[Q_n] \leftarrow (c, x, h_n)$ and returns h to S.

Random oracle query $\mathrm{G}_{\pi}(h)$**:** If $GT[\pi, h]$ is not defined, then A chooses $GT[\pi, h] \stackrel{\$}{\leftarrow} \mathbb{G}_{\pi}$. It returns $GT[\pi, h]$ to F.

Signing query $\mathsf{Sign}(\pi^{*-1}, M_n, \Sigma_{n-1})$ **:** Parse Σ_{n-1} as $(\vec{\pi}, m_{n-1}, X_{n-1}, h_{n-1})$. Algorithm A looks up the entry $HT[\vec{\pi}|\pi^*\|M_n\|m_{n-1}\|X_{n-1}] = (c, X_n, h_n)$, which must exist by Property 4 of a sequential forger. Let $(m_n, \mu_n) = \mathsf{enc}_{\pi^*}(M_n\|m_{n-1}\|X_{n-1})$. If $c = 0$ then A returns $\Sigma_n = (\vec{\pi}\|\pi^*, m_n, X_n, h_n)$. If $c = 1$, then we say that event $\textsc{Bad}_2$ occurred and A aborts.

At the end of its execution, the forger outputs its forgery $\Sigma_n = (\vec{\pi}, m_n, X_n, h_n)$. By Property 5 the forgery is valid, so $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_n) = (\vec{\pi}, \vec{M})$ and there exists an index $1 \leq i \leq n$ such that $\pi_i = \pi^*$ and S never made a query $\mathsf{Sign}(\pi^{*-1}, M_i, \Sigma_{i-1})$ for the unique tuple Σ_{i-1} such that $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_{i-1}) = (\vec{\pi}|_{i-1}, \vec{M}|_{i-1})$. Let m_{i-1}, X_{i-1}, μ_i, and X_i be the intermediate values obtained during the computation of $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_n)$.

Algorithm A looks up $HT[\vec{\pi}|_i\|M_i\|m_{i-1}\|X_{i-1}] = (c, y, h_i)$ and $GT[\pi^*, h_i] = g_i$. (We know that these entries are defined by Property 5 of a sequential forger.) If $c = 0$ then we say that event $\textsc{Bad}_2$ occurred and A aborts. If $c = 1$ then we have that $\rho^*(y) = g_i + \mu_i$, but since Σ_n is valid we also have that $\pi^*(X_i) = g_i + \mu_i$. Since $\mathbb{G}_{\pi^*}$ is a group we therefore have that $\pi^*(X_i) = \rho^*(y)$; algorithm A outputs (X_i, y) as the claw for (π^*, ρ^*).

We now use Coron's technique [Cor00] for a detailed analysis of A's success probability. Algorithm A wins the game if forger S wins and neither of $\textsc{Bad}_1$ and $\textsc{Bad}_2$ occurs, i.e.

$$\begin{aligned}\Pr[\,\mathsf{A}\text{ wins}\,] &= \Pr\left[\,\mathsf{S}\text{ wins} \wedge \overline{\textsc{Bad}_1} \wedge \overline{\textsc{Bad}_2}\,\right] \\ &\geq \Pr\left[\,\mathsf{S}\text{ wins} \mid \overline{\textsc{Bad}_1} \wedge \overline{\textsc{Bad}_2}\,\right] \cdot \Pr\left[\,\overline{\textsc{Bad}_2}\,\right] - \Pr\left[\,\overline{\textsc{Bad}_1}\,\right]\end{aligned}$$

In the event $\overline{\textsc{Bad}_1} \wedge \overline{\textsc{Bad}_2}$ the environment of S is distributed exactly as in a real attack, so

$$\Pr\left[\,\mathsf{S}\text{ wins} \mid \overline{\textsc{Bad}_1} \wedge \overline{\textsc{Bad}_2}\,\right] \;=\; \epsilon\ .$$

Event $\textsc{Bad}_1$ occurs only if $h_n = h \oplus h_{n-1}$ already occurred in $GT[\cdot,\cdot]$, where h is a fresh random ℓ-bit string. The probability that this happens is therefore the number of entries in $GT[\cdot,\cdot]$ divided by 2^ℓ. Summing over all q_{H} queries to $\mathrm{H}(\cdot)$ we have

$$\Pr[\,\textsc{Bad}_1\,] \;\leq\; \frac{q_{\mathrm{H}}(q_{\mathrm{H}}+q_{\mathrm{G}})}{2^\ell}\ .$$

The probability that event $\overline{\textsc{Bad}_2}$ occurs is

$$\Pr\left[\,\overline{\textsc{Bad}_2}\,\right] \;\geq\; \delta^{q_{\mathrm{S}}}(1-\delta)\ .$$

This probability is maximal for $\delta = 1/(q_{\mathrm{S}}+1)$. Filling in this value gives

$$\Pr\left[\,\overline{\textsc{Bad}_2}\,\right] \;\geq\; \frac{1}{e(q_{\mathrm{S}}+1)}\ ,$$

so that the success probability of A in finding a claw for (π^*, ρ^*) is

$$\epsilon' \;\geq\; \frac{\epsilon}{e(q_{\mathrm{S}}+1)} - \frac{q_{\mathrm{H}}(q_{\mathrm{H}}+q_{\mathrm{G}})}{2^\ell}\ .$$

To estimate the running time of A, we ignore all costs other than permutation evaluations. Each $\mathrm{H}(\cdot)$ query explicitly induces $1/d$ permutation evaluations on average, but in fact hides an additional evaluation to maintain the graph structure that allows A to find the sequence $(Q_1, \ldots, Q_{n-1})$ (see the proof of Lemma 5.3). The verification of the forgery takes up to $n_{\max}$ additional permutations. Overall, we have that $t' \leq t + ((1/d+1)q_{\mathrm{H}} + n_{\max}) \cdot t_\pi$. ∎

We can now shed some more technical light on how our security proof, unlike that of the $\mathcal{LMRS}$ scheme, does not rely on the fact that the permutation family be certified. The $\mathcal{LMRS}$ scheme involves a full-domain random oracle that essentially plays the combined role of our $\mathrm{H}(\cdot)$ and $\mathrm{G}.(\cdot)$ oracles. In their proof, however, the responses of this oracle need to be simulated such that the claw-finding algorithm A knows a related preimage for *all* permutations π, rather than just the target permutation π^*. The standard way to do this is to choose a random preimage $x \xleftarrow{\$} D_\pi$ and compute the random oracle output as $\pi(x)$. This output is only correctly distributed, however, if π really is a permutation, hence the requirement that Π be certified. In our proof, the algorithm A only needs to know preimages related to queries $\mathrm{G}_{\pi^*}(\cdot)$, and can therefore sample random elements from $\mathbb{G}_\pi$ directly to simulate responses for $\mathrm{G}.(\cdot)$ with $\pi \neq \pi^*$.

6 Variations on the Main Construction

Sequential aggregate signatures. If the message recovery functionality of our $\mathcal{SASD}$ scheme is undesirable, then the following "purebred" sequential aggregate signature scheme $\mathcal{SAS}$ is easily derived from our $\mathcal{SASD}$ scheme. The signer's public and private key are again a permutation π and its inverse π^{-1}; aggregate signing and verification are as follows:

Algorithm $\mathsf{Sign}^{\mathrm{H,G}}(\pi_n^{-1}, M_n, \vec{\pi}, \vec{M}, \sigma_{n-1})$:	Algorithm $\mathsf{Vf}^{\mathrm{H,G}}(\vec{\pi}, \vec{M}, \sigma_n)$:
If $n = \lvert\vec{\pi}\rvert = 1$ then $\sigma_0 \leftarrow (\varepsilon, \varepsilon, 0^\ell)$	Parse σ_n as (x_n, X_n, h_n) where $n = \lvert\vec{\pi}\rvert$
Parse σ_{n-1} as $(x_{n-1}, X_{n-1}, h_{n-1})$	For $i = n, \ldots, 1$ do
If $\mathsf{Vf}^{\mathrm{H,G}}(\vec{\pi}, \vec{M}, \sigma_{n-1}) = 0$ then return $\perp$	If $\lvert\mathbb{G}_{\pi_i}\rvert < 2^\ell$ then return 0
$(x_n, \xi_n) \leftarrow \mathsf{enc}_{\pi_n}(x_{n-1} \| X_{n-1})$	$g_i \leftarrow \mathrm{G}_{\pi_i}(h_i)$; $\xi_i \leftarrow \pi_i(X_i) - g_i$
$h_n \leftarrow h_{n-1} \oplus \mathrm{H}(\vec{\pi}\lvert\pi_n\|\vec{M}\lvert M_n\|x_{n-1}\|X_{n-1})$	$x_{i-1}\|X_{i-1} \leftarrow \mathsf{dec}_{\pi_i}(m_i, \mu_i)$
$g_n \leftarrow \mathrm{G}_{\pi_n}(h_n)$	$h_{i-1} \leftarrow h_i \oplus \mathrm{H}(\vec{\pi}\lvert_i\|\vec{M}\lvert_i\|x_{i-1}\|X_{i-1})$
$X_n \leftarrow \pi_n^{-1}(g_n + \xi_n)$	If $(x_0, X_0, h_0) = (\varepsilon, \varepsilon, 0^\ell)$ then return 1
Return $\sigma_n \leftarrow (x_n, X_n, h_n)$	Else return 0.

Just like the $\mathcal{SASD}$ scheme, the $\mathcal{SAS}$ scheme allows for efficient instantiations based on low-exponent RSA and factoring, and allows signers to independently choose their security parameter k. The signature size depends on various issues such as the used encoding and permutation family, but for the instantiation based on RSA with $e = 3$ and moduli of size at most k_f, the signature will be at most $k_\mathrm{f} + \ell + n$ bits long. Signing and verification take the same effort as for the $\mathcal{SASD}$ scheme.

The scheme can be proved secure in the random oracle model under the standard (non-KOSK) security notion of [LMRS04, BNN07]. The proof is almost identical to that of Theorem 5.1; the definition of a sequential forger needs to be adapted to queries of the form $\mathrm{H}(\vec{\pi}\|\vec{M}\|x_{n-1}\|X_{n-1})$, and nodes in the graph $\mathcal{G}$ will be identified by tuples $(\vec{\pi}, \vec{M}, x, Y)$. The concrete security bounds are identical to those obtained in Theorem 5.1.

Achieving tight security. Closer inspection of Theorem 5.1 learns that the reduction loses a factor q_S in the success probability of the claw-finding algorithm A. In principle, this means that higher security parameters have to be used in order to achieve the same security level, thereby increasing the length of keys and signatures. One can apply the techniques of Katz-Wang [KW03] however to obtain a scheme $\mathcal{SASDt}$ with a tight security reduction, at the minimal cost of an increase in signature length of n bits. (The same techniques have also been applied to achieve tight security for the $\mathcal{LMRS}$ scheme in [BNN07].) Key generation is as for the $\mathcal{SASD}$ scheme, signing and verification are as follows:

Algorithm $\mathsf{Sign}^{\mathrm{H,G}}(\pi^{-1}, M_n, \Sigma_{n-1})$:
- If $ST[M_n, \Sigma_{n-1}] = \Sigma_n \neq \varepsilon$ then return Σ_n
- If $n = 1$ then $\Sigma_0 \leftarrow (\vec{\varepsilon}, \vec{\varepsilon}, \varepsilon, \varepsilon, 0^\ell)$
- Parse Σ_{n-1} as $(\vec{b}, \vec{\pi}, m_{n-1}, X_{n-1}, h_{n-1})$
- If $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_{n-1}) = (\perp, \perp)$ then return $\perp$
- $b_n \xleftarrow{\$} \{0, 1\}$
- $(m_n, \mu_n) \leftarrow \mathsf{enc}_{\pi_n}(M_n \| m_{n-1} \| X_{n-1})$
- $h_n \leftarrow h_{n-1} \oplus \mathrm{H}(\vec{b}|b_n \| \vec{\pi}|\pi_n \| M_n \| m_{n-1} \| X_{n-1})$
- $g_n \leftarrow \mathrm{G}_{\pi_n}(h_n)$; $X_n \leftarrow \pi_n^{-1}(g_n + \mu_n)$
- $ST[M_n, \Sigma_{n-1}] \leftarrow (\vec{b} \| b_n, \vec{\pi} \| \pi_n, m_n, X_n, h_n)$
- Return $ST[M_n, \Sigma_{n-1}]$

Algorithm $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma)$:
- Parse Σ as $(\vec{b}, \vec{\pi}, m_n, X_n, h_n)$ where $n = |\vec{b}| = |\vec{\pi}|$
- For $i = n, \ldots, 1$ do
  - If $|\mathbb{G}_{\pi_i}| < 2^\ell$ then return $(\perp, \perp)$
  - $g_i \leftarrow \mathrm{G}_{\pi_i}(h_i)$; $\mu_i \leftarrow \pi_i(X_i) - g_i$
  - $M_i \| m_{i-1} \| X_{i-1} \leftarrow \mathsf{dec}_{\pi_i}(m_i, \mu_i)$
  - $h_{i-1} \leftarrow h_i \oplus \mathrm{H}(\vec{b}|_i, \vec{\pi}|_i, M_i, m_{i-1}, X_{i-1})$
- If $(m_0, X_0, h_0) = (\varepsilon, \varepsilon, 0^\ell)$
- Then return $\big(\vec{\pi}, \vec{M} = (M_1, \ldots, M_n)\big)$
- Else return $(\perp, \perp)$.

The signing algorithm described here is stateful, but as already noted in [KW03] it can easily be made stateless by using an additional random oracle $\mathrm{F} : \{0,1\}^* \to \{0,1\}$ to generate b_n as $\mathrm{F}(\pi^{-1} \| M_n \| \Sigma_{n-1})$. Alternatively, one can generate b_n using a pseudo-random function $\mathrm{F}(K, M_n \| \Sigma_{n-1})$ and include the key K in the signing key *sk*. In the latter case the security of the scheme additionally relies on the pseudo-randomness of F, of course. The proof of the following theorem is an adaptation of the proof of Theorem 5.1 using techniques from [KW03, BNN07]. For completeness it is given in detail in Appendix A.

Theorem 6.1 If there exists a forger F that $(t, q_{\mathrm{S}}, q_{\mathrm{H}}, q_{\mathrm{G}}, n_{\max}, \epsilon)$-breaks $\mathcal{SASD}t$ in the random oracle model, then there exists a claw-finding algorithm A that (t', ϵ')-breaks Π with

$$\begin{aligned} \epsilon' &\geq \frac{\epsilon}{2} - \frac{7\big(q_{\mathrm{H}} + q_{\mathrm{G}} + 2n_{\max}(q_{\mathrm{S}} + 1)\big)^2}{2^\ell} \\ t' &\leq t + (2/d + 4)(q_{\mathrm{H}} + n_{\max}(q_{\mathrm{S}} + 1)) \cdot t_\pi \ . \end{aligned}$$

7 Non-Interactive Multi-Signed Data

When all signers are authenticating the same message M, a more efficient scheme exists that does not require any interaction among the signers at all (as opposed to the sequential interaction required for the other schemes in this paper). Here, all signers independently generate their signature shares, which can then be combined by any third party into the final signature.

Syntax and security. A *multi-signed data* (MSD) scheme is a tuple of algorithms $\mathcal{MSD} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Comb}, \mathsf{Vf})$. A signer generates his own key pair via $(pk, sk) \xleftarrow{\$} \mathsf{Kg}$. Each signer creates a partial signature on M via $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, M)$. Any third party can combine a list of partial signatures $\vec{\sigma}$ into the final signed data via $\Sigma \xleftarrow{\$} \mathsf{Comb}(\vec{pk}, M, \vec{\sigma})$. The verification algorithm $\mathsf{Vf}(\Sigma)$ returns $(\vec{pk}, M)$ to indicate that Σ is valid for signers $\vec{pk}$ and message M, or returns $(\perp, \perp)$ to indicate rejection. Correctness requires that $\mathsf{Vf}(\Sigma) = (pk, M)$ with probability one for all messages M if all signers behave honestly.

In the experiment defining security, the forger F is given a freshly generated public key pk^* as input, and has access to a signing oracle $\mathsf{Sign}(sk^*, \cdot)$ for the corresponding secret key sk^*. It wins if it can output a forgery Σ such that $\mathsf{Vf}(\Sigma) = (\vec{pk}, M) \neq (\perp, \perp)$ with $pk_i = pk^*$ for some $1 \leq i \leq |\vec{pk}|$ and F never queried M to the signing oracle. We say that F $(t, q_{\mathrm{S}}, n_{\max}, \epsilon)$-breaks $\mathcal{MSD}$ if it runs in time at most t, makes at most q_{S} signing queries, its forgery contains at most $n_{\max}$ signatures, and wins the above game with probability at least ϵ. In the random oracle model, we additionally bound the maximum number of queries that F can make to each random oracle separately.

THE SCHEME. Let $k, \ell \in \mathbb{N}$ be security parameters where k is chosen by each signer independently and ℓ is fixed system-wide. Let Π be a family of claw-free trapdoor permutations, let $\mathbb{G}_\pi \subseteq D_\pi$ for $\pi \in \Pi$ be a group, and let $\mathrm{H} : \{0,1\}^* \rightarrow \{0,1\}^\ell$ and $\mathrm{G}_\pi : \{0,1\}^\ell \rightarrow \mathbb{G}_\pi$ be random oracles, exactly as for the $\mathcal{SASD}$ scheme. The encoding and decoding functions are different though: we assume two separate encoding algorithms $\mathsf{enc}_\pi : \{0,1\}^* \rightarrow \mathbb{G}_\pi$, $\mathsf{enc}_{(\pi_1,\ldots,\pi_n)} : \{0,1\}^* \rightarrow \{0,1\}^*$, and a decoding algorithm $\mathsf{dec}_{(\pi_1,\ldots,\pi_n)} : \{0,1\}^* \times \mathbb{G}_{\pi_1} \times \ldots \times \mathbb{G}_{\pi_n}$ such that $\mathsf{enc}_{\pi_i}(M)$ outputs a group element $\mu_i \in \mathbb{G}_{\pi_i}$; $\mathsf{enc}_{\vec{\pi}}(M)$ outputs a partial message m; and the injective function $\mathsf{dec}_{\vec{\pi}}(m, \vec{\mu})$ reconstructs the original message M. Key generation consists again of generating a random permutation π as the public key and its inverse π^{-1} as secret key; the signing, combining, and verification algorithms are described below.

Algorithm $\mathsf{Sign}^{\mathrm{H,G}}(\pi_n^{-1}, M)$:	Algorithm $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma)$:
$\mu \leftarrow \mathsf{enc}_\pi(M)$; $h \leftarrow \mathrm{H}(M)$	Parse Σ as $(\vec{\pi}, m, \vec{X}, h)$; $n \leftarrow \lvert\vec{\pi}\rvert$
$g \leftarrow \mathrm{G}_\pi(h)$; $X \leftarrow \pi^{-1}(\mu + g)$	If $\lvert\vec{X}\rvert \neq n$ then return $(\perp, \perp)$
Return X	For $i = 1, \ldots, n$ do
Algorithm $\mathsf{Comb}^{\mathrm{H,G}}(\vec{\pi}, M, \vec{X})$:	$g_i \leftarrow \mathrm{G}_{\pi_i}(h_i)$; $\mu_i \leftarrow \pi_i(X_i) - g_i$
$m \leftarrow \mathsf{enc}_{\vec{\pi}}(M)$; $h \leftarrow \mathrm{H}(M)$	$M \leftarrow \mathsf{dec}_{\vec{\pi}}\big(m, (\mu_1, \ldots, \mu_n)\big)$
Return $\Sigma \leftarrow (\vec{\pi}, m, \vec{X}, h)$	If $\mathrm{H}(M) = h$ then return $(\vec{\pi}, M)$
	Else return $(\perp, \perp)$.

SECURITY. We prove the scheme secure in the random oracle model under the claw-freeness of the permutation family Π. The proof reuses techniques from [BR96, Cor00] and is given below. The proof loses a factor q_S in the reduction; a tight variant can be constructed using the techniques of Katz-Wang [KW03].

Theorem 7.1 If there exists a forger F that $(t, q_\mathrm{S}, q_\mathrm{H}, q_\mathrm{G}, \epsilon)$-breaks $\mathcal{MSD}$ in the random oracle model, then there exists a claw-finding algorithm A that (t', ϵ')-breaks Π claw-free for

$$\begin{aligned} \epsilon' &\geq \frac{\epsilon}{e(q_\mathrm{S}+1)} - \frac{(q_\mathrm{G}+q_\mathrm{H}+q_\mathrm{S}+n_{\max}+1)^2}{2^\ell} \\ t' &\leq t + \frac{q_\mathrm{H}+q_\mathrm{S}+n_{\max}+1}{d} \cdot t_\pi \ . \end{aligned}$$

Proof: Given a forger F, consider the following claw-finding algorithm A. On input target permutation π^*, A runs F on input π^*, answering its oracle queries as follows while keeping associative tables $HT[\cdot]$, $GT[\cdot,\cdot]$.

Random oracle query $\mathrm{H}(M)$**:** If $HT[M] = (c, x, h)$ is defined, then it returns h. Otherwise, it chooses $h \stackrel{\$}{\leftarrow} \{0,1\}^\ell$. If $GT[\pi^*, h]$ is already defined, then we say that event BAD_1 occurred and algorithm A aborts. Otherwise, it chooses $c \stackrel{\delta}{\leftarrow} \{0,1\}$ and computes $\mu \leftarrow \mathsf{enc}_{\pi^*}(M)$. If $c = 0$ then A repeatedly chooses $x \stackrel{\$}{\leftarrow} D_{\pi^*}$ and computes $g \leftarrow \pi^*(x) - \mu$ until $g \in \mathbb{G}_{\pi^*}$. If $c = 1$ then it repeatedly chooses $x \stackrel{\$}{\leftarrow} D_{\pi^*}$ and computes $g \leftarrow \rho^*(x) - \mu$ until $g \in \mathbb{G}_{\pi^*}$. Algorithm A stores $GT[\pi^*, h] \leftarrow g$ and $HT[M] \leftarrow (c, x, h)$, and returns h to F.

Random oracle query $\mathrm{G}_\pi(h)$**:** If $GT[\pi, h]$ is not defined, then A chooses $GT[\pi, h] \stackrel{\$}{\leftarrow} \mathbb{G}_\pi$. It returns $GT[\pi, h]$ to F.

Signing query $\mathsf{Sign}(M)$**:** First, A simulates an additional query $\mathrm{H}(M)$ to ensure that entries $HT[M] = (c, X, h)$ and $GT[\pi^*, h] = g$ have been defined. If $c = 1$ then we say that event BAD_2 occurred and A aborts. If $c = 0$ then it returns X as the partial signature.

Eventually, F outputs its forgery $\Sigma = (\vec{\pi}, m, \vec{X}, h)$. Algorithm A verifies that the forgery is valid, meaning $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma) = (\vec{\pi}, M) \neq (\perp, \perp)$ where $\mathrm{H}(\cdot)$ and $\mathrm{G}.(\cdot)$ are simulated as above. Let $1 \leq i \leq |\vec{\pi}|$ be the index such that $\pi_i = \pi^*$, and let $\mu_i \leftarrow \mathsf{enc}_{\pi^*}(M)$. Algorithm A looks up $HT[M] = (c, y, h)$ and $GT[\pi, h] = g$. If $c = 0$ then we say that event BAD_2 occurred and A aborts. Otherwise, since $\mathsf{dec}_{\vec{\pi}}$ is injective, we have that $\pi^*(X_i) - g = \rho^*(y) - g = \mu_i$, so $\pi^*(X_i) = \rho^*(y)$. Algorithm A outputs (X_i, y) as its claw for (π^*, ρ^*).

It is clear that A is successful in finding a claw if F succeeds in creating a valid forgery and A doesn't abort, meaning

$$\begin{aligned}\epsilon' &\geq \Pr\left[\,\mathsf{A}\text{ wins} \wedge \overline{\mathrm{BAD}_1} \wedge \overline{\mathrm{BAD}_2}\,\right] \\ &\geq \Pr\left[\,\mathsf{A}\text{ wins} \mid \overline{\mathrm{BAD}_2} \wedge \overline{\mathrm{BAD}_1}\,\right] \cdot \Pr\left[\,\overline{\mathrm{BAD}_2}\,\right] - \Pr\left[\,\mathrm{BAD}_1\,\right] . \qquad (4)\end{aligned}$$

It is also clear that A's simulation of F's environment is perfect as long as it doesn't abort, meaning

$$\Pr\left[\,\mathsf{A}\text{ wins} \mid \overline{\mathrm{BAD}_2} \wedge \overline{\mathrm{BAD}_1}\,\right] \;\geq\; \epsilon \qquad (5)$$

Event BAD_1 occurs when a randomly chosen value $h \xleftarrow{\$} \{0,1\}^\ell$ already appeared in $GT[\cdot,\cdot]$ as an entry $GT[\pi, h]$. This happens with a probability that is the number of entries in $GT[\cdot,\cdot]$ divided by 2^ℓ, or at most $(q_{\mathrm{H}} + q_{\mathrm{G}} + q_{\mathrm{S}} + n_{\max})/2^\ell$. Summing over all $(q_{\mathrm{H}} + q_{\mathrm{S}} + 1)$ calls to the simulation of $\mathrm{H}(\cdot)$ gives

$$\Pr\left[\,\mathrm{BAD}_1\,\right] \;\leq\; \frac{(q_{\mathrm{H}} + q_{\mathrm{G}} + q_{\mathrm{S}} + n_{\max} + 1)^2}{2^\ell} . \qquad (6)$$

The event $\overline{\mathrm{BAD}_2}$ occurs if $c = 0$ during the simulation of all signature queries and $c = 1$ for the forgery. In the event $\overline{\mathrm{BAD}_1}$ the value of h used in the forgery must be different from any of the ones involved in any of the signing queries, so we have that

$$\Pr\left[\,\overline{\mathrm{BAD}_2} \mid \overline{\mathrm{BAD}_1}\,\right] \;=\; \delta^{q_{\mathrm{S}}} \cdot (1 - \delta) . \qquad (7)$$

This expression reaches a minimum for $\delta = 1 - 1/(q_{\mathrm{S}} + 1)$, where we have that

$$\delta^{q_{\mathrm{S}}} \cdot (1-\delta) \;=\; \left(1 - \frac{1}{q_{\mathrm{S}}+1}\right)^{q_{\mathrm{S}}} \cdot \frac{1}{q_{\mathrm{S}}+1} \;>\; \left(1 - \frac{1}{q_{\mathrm{S}}}\right)^{q_{\mathrm{S}}} \cdot \frac{1}{q_{\mathrm{S}}+1} \;>\; \frac{1}{e(q_{\mathrm{S}}+1)} .$$

Putting together Equations (4), (5), (6), (7), and (7) yields the bound on ϵ' in Theorem 7.1. Ignoring the time needed for all computations other than permutation evaluations, the bound on the running time t' of A in Theorem 7.1 is clear from the construction of A. ■

INSTANTIATIONS. One can obtain instantiations of $\mathcal{MSD}$ from low-exponent RSA and factoring using the same permutation families and group structures described in Section 4. For the encoding function, one could for example split the message in $k_{\max}$-bit blocks (e.g. $k_{\max} = 4096$ when using RSA) and let μ be the first k bits of the block with index $\mathsf{h}(\pi, M)$ where $\mathsf{h} : \{0,1\}^* \to \{1, \ldots, \lfloor |M|/k_{\max} \rfloor\}$ is a non-cryptographic hash function. The function $\mathsf{enc}_{\vec{\pi}}(M)$ returns the remaining bits of M; decoding works by reconcatenating the different message parts in the correct order. For long enough messages M (in particular, $|M| \gg nk_{\max}$), there is no overlap between the message parts of different co-signers, and $\mathcal{MSD}$ achieves the promised length savings.

Alternatively, if the list of co-signers is known at the time of signing, one could modify the scheme so that encoding is more effective for short messages. Namely, one could use a single encoding function $\mathsf{enc}_{(\pi_1,\ldots,\pi_n)} : \{0,1\}^* \to \{0,1\}^* \times \mathbb{G}_{\pi_1} \times \ldots \times \mathbb{G}_{\pi_n}$ that ensures there is no overlap between the different message parts. In this case, however, the scheme needs to be modified to include $\vec{\pi}$ in the computation of $h \leftarrow \mathrm{H}(\vec{\pi}, M)$, because otherwise there may exist (contrived) encoding algorithms that render the scheme insecure.

To explain the efficiency estimates in Table 1, we assume an RSA-based instantiation with $e = 3$ and $\mathbb{G}_{N,e} = 0\|\{0,1\}^{k_\mathrm{f}-1}$ with the XOR operation. The net bandwidth overhead depends on the length of the message and, related to that, on the number of "collisions" between the recovered message parts for different signers. In the best case, i.e., $|M| \geq n \cdot k_\mathrm{f}$ and no collisions, the net overhead is $\ell + n$ bits. In the worst case, i.e., $|M| = 0$, the overhead is $nk_\mathrm{f} + \ell$. Signing involves a single modular exponentiation, while verification takes n evaluations of π, which can be performed by $2n$ modular multiplications.

we consider the instantiation with $e = 3$ and $\mathbb{G}_{N,e} = \{0\|x \;:\; x \in \{0,1\}^{k_\mathrm{f}-1}\}$ with the XOR operation, and we assume that signers use security parameter k_f or smaller. The exact bandwidth overhead depends on the amount of "recoverable" data μ_i, but varies between ℓ bits in the best case and $\ell + k_\mathrm{f} + n$ bits in the worst case. Typical bandwidth overhead will be around $\ell + n$ bits (for n signers using the same security parameter k_f and $|M_1| \geq k_\mathrm{f}$). The most expensive operations in signing and verification are the computations of π and π^{-1}, which for $e = 3$ take two modular multiplications and one modular exponentiation each, respectively. Ignoring cheap operations such as XORs and hash function evaluations, verifying aggregate signed data by n signers takes $2n$ modular multiplications. Placing a signature on top of an aggregate of n signers takes $2n$ multiplications (to verify the existing aggregate) and one exponentiation.

Acknowledgements

The author would like to thank the anonymous referees of EUROCRYPT 2008 and IEEE Transactions on Information Theory for their helpful suggestions. This work was supported in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II, in part by a postdoctoral fellowship from the Research Foundation – Flanders (FWO), and in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

References

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *20th Annual ACM Symposium on Theory of Computing*, pages 103–112. ACM Press, 1988. (Cited on page 2.)

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer-Verlag, 2003. (Cited on pages 1, 2 and 3.)

[BGOY07] Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In *ACM CCS 07: 14th Conference on Computer and Communications Security*, pages 276–285. ACM Press, 2007. (Cited on page 2.)

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *ACM CCS 06: 13th Conference on Computer and Communications Security*, pages 390–399. ACM Press, 2006. (Cited on page 3.)

[BNN07] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *ICALP 2007: 34th International Colloquium on Automata, Languages and Programming*, volume 4596 of *Lecture Notes in Computer Science*, pages 411–422. Springer-Verlag, 2007. Full version available as Cryptology ePrint Archive, Report 2006/285. (Cited on pages 3, 4, 13, 16 and 17.)

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer-Verlag, 2003. (Cited on page 3.)

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73. ACM Press, 1993. (Cited on page 4.)

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer-Verlag, 1996. (Cited on pages 2, 5, 13 and 18.)

[BY96] Mihir Bellare and Moti Yung. Certifying permutations: Noninteractive zero-knowledge based on any trapdoor permutation. *Journal of Cryptology*, 9(3):149–166, 1996. (Cited on page 2.)

[CM99] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 107–122. Springer-Verlag, 1999. (Cited on page 2.)

[Cor00] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer-Verlag, 2000. (Cited on pages 13, 15 and 18.)

[CPP07] Dario Catalano, David Pointcheval, and Thomas Pornin. Trapdoor hard-to-invert group isomorphisms and their application to password-based authentication. *Journal of Cryptology*, 20(1):115–149, 2007. (Cited on page 2.)

[GR06] Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, volume

3958 of *Lecture Notes in Computer Science*, pages 257–273. Springer-Verlag, 2006. (Cited on page 2.)

[HOT04] Ryotaro Hayashi, Tatsuaki Okamoto, and Keisuke Tanaka. An RSA family of trapdoor permutations with a common domain and its applications. In Feng Bao, Robert Deng, and Jianying Zhou, editors, *PKC 2004: 7th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2947 of *Lecture Notes in Computer Science*, pages 291–304. Springer-Verlag, 2004. (Cited on page 7.)

[IN83] K. Itakura and K. Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, 71:1–8, 1983. (Cited on page 3.)

[KLS00] Stephen Kent, Charles Lynn, and Karen Seo. Secure border gateway protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, April 2000. (Cited on page 1.)

[KW03] Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In *ACM CCS 03: 10th Conference on Computer and Communications Security*, pages 155–164. ACM Press, 2003. (Cited on pages 16, 17 and 18.)

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 74–90. Springer-Verlag, 2004. (Cited on pages 1, 2, 3, 4, 13 and 16.)

[LOS+06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 465–485. Springer-Verlag, 2006. (Cited on pages 2 and 3.)

[Nev08] Gregory Neven. Efficient sequential aggregate signed data. In Nigel Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 52–69. Springer-Verlag, 2008. (Cited on page 1.)

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978. (Cited on page 7.)

A Proof of Theorem 6.1

As for the proof of Theorem 5.1, we will do the proof of Theorem 6.1 in two steps. First, we modify the definition of a sequential forger of Definition 5.2 to queries of the form $\mathrm{H}(\vec{b}, \vec{\pi}, M_n, m_{n-1}, X_{n-1})$, and we show that for any forger F there exists a sequential forger S achieving the same bounds as those stated in Lemma 5.3 for the $\mathcal{SASD}$ scheme. The proof is a straightforward adaptation of that of Lemma 5.3, and is omitted here. Theorem 6.1 follows easily from (the adaptation of) Lemma 5.3 and the following lemma.

Lemma A.1 If there exists a sequential forger S that $(t, q_{\mathrm{S}}, q_{\mathrm{H}}, q_{\mathrm{G}}, n_{\max}, \epsilon)$-breaks $\mathcal{SASDt}$, then there exists a claw-finding algorithm A that (t', ϵ')-breaks Π for

$$\begin{aligned} \epsilon' &\geq \frac{\epsilon}{2} - \frac{2q_{\mathrm{H}}(2q_{\mathrm{H}} + q_{\mathrm{G}})}{2^{\ell}} \\ t' &\leq t + ((2/d+1)q_{\mathrm{H}} + n_{\max}) \cdot t_{\pi} \ . \end{aligned}$$

Proof: Given a sequential forger S against $\mathcal{SASDt}$, we build a claw-finding algorithm A against Π. Algorithm A maintains initially empty associative arrays $HT[\cdot]$ and $GT[\cdot,\cdot]$. On input π^*, ρ^*, algorithm A runs S on target public key π^*, and responds to its oracle queries as follows:

Random oracle query $\mathrm{H}(Q_n)$**:** Parse Q_n as $\vec{b}\|\vec{\pi}\|M_n\|m_{n-1}\|X_{n-1}$ where $n = |\vec{b}| = |\vec{\pi}|$. If $n > 1$, then A finds the unique sequence of queries $(Q_1, \ldots, Q_{n-1})$ as per Property 3 of a sequential forger, and looks up $HT[Q_{n-1}] = (c, x, h, h_{n-1})$. If $n = 1$ it sets $h_0 \leftarrow 0^{\ell}$.

If $\pi_n \neq \pi^*$ then A chooses $h \xleftarrow{\$} \{0,1\}^{\ell}$, computes $h_n \leftarrow h \oplus h_{n-1}$, stores $HT[Q_n] \leftarrow (\perp, \perp, h, h_n)$, and returns h to S.

If $\pi_n = \pi^*$ then A chooses $c \xleftarrow{\$} \{0,1\}$ and $h^{(0)}, h^{(1)} \xleftarrow{\$} \{0,1\}^{\ell}$, and computes $(m_n, \mu_n) = \mathsf{enc}_{\pi^*}(M_n\|m_{n-1}\|X_{n-1})$, $h_n^{(0)} \leftarrow h^{(0)} \oplus h_{n-1}$, and $h_n^{(1)} \leftarrow h^{(1)} \oplus h_{n-1}$. Let $Q_n^{(\beta)} = \vec{b}|_{n-1}|\beta\| \vec{\pi}\|M_n\|m_{n-1}\|X_{n-1}$ for $\beta \in \{0,1\}$. Note that $Q_n^{(b_n)} = Q_n$.

Algorithm A repeatedly chooses $x^{(c)} \xleftarrow{\$} D_{\pi^*}$ and computes $g_n^{(c)} \leftarrow \pi^*(x^{(c)}) - \mu_n$ until $g_n^{(c)} \in \mathbb{G}_{\pi^*}$. It then repeatedly chooses $x^{(1-c)} \xleftarrow{\$} D_{\pi^*}$ and computes $g_n^{(1-c)} \leftarrow \rho^*(x^{(1-c)}) - \mu_n$ until $g_n^{(1-c)} \in \mathbb{G}_{\pi^*}$. If one of $GT[\pi^*, h_n^{(0)}]$ or $GT[\pi^*, h_n^{(1)}]$ is already defined, then we say that event BAD_1 occurred and A aborts; otherwise, it stores $HT[Q_n^{(0)}] \leftarrow (c, x^{(0)}, h^{(0)}, h_n^{(0)})$, $HT[Q_n^{(1)}] \leftarrow (c, x^{(1)}, h^{(1)}, h_n^{(1)})$, $GT[\pi^*, h_n^{(0)}] \leftarrow g_n^{(0)}$, and $GT[\pi^*, h_n^{(1)}] \leftarrow g_n^{(1)}$. It returns the value $h^{(b_n)}$ to S.

Random oracle query $\mathrm{G}_{\pi}(h)$**:** If $GT[\pi, h]$ is not defined, then A chooses $GT[\pi, h] \xleftarrow{\$} \mathbb{G}_{\pi}$. It returns $GT[\pi, h]$ to F.

Signing query $\mathsf{Sign}(\pi^{*-1}, M_n, \Sigma_{n-1})$: Parse Σ_{n-1} as $(\vec{b}, \vec{\pi}, m_{n-1}, X_{n-1}, h_{n-1})$ where $n = |\vec{b}| + 1$. Algorithm A looks up the entries $HT[\vec{b}|\beta\|\vec{\pi}|\pi^*\|M_n\|m_{n-1}\|X_{n-1}] = (c, X_n^{(\beta)}, h^{(\beta)}, h_n^{(\beta)})$ for $\beta \in \{0,1\}$, which must exist due to Property 4 of a sequential forger. Let $(m_n, \mu_n) = \mathsf{enc}_{\pi^*}(M_n\|m_{n-1}\|X_{n-1})$. Algorithm A returns signed data $\Sigma_n = \big(\vec{b}\|c, \vec{\pi}\|\pi^*, m_n, X_n^{(c)}, h_n^{(c)}\big)$.

At the end of its execution, the forger outputs its forgery $\Sigma_n = (\vec{b}, \vec{\pi}, m_n, X_n, h_n)$ where $n = |\vec{b}|$. By Property 5 the forgery is valid, so $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_n) = (\vec{\pi}, \vec{M})$ and there exists an index $1 \leq i \leq n$ such that $\pi_i = \pi^*$ and S never made a query $\mathsf{Sign}(\pi^{*-1}, M_i, \Sigma_{i-1})$ for any tuple Σ_{i-1} such that $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_{i-1}) = (\vec{\pi}|_{i-1}, \vec{M}|_{i-1})$. Let m_{i-1}, X_{i-1}, μ_i, and X_i be the intermediate values obtained during the evaluation of $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_n)$.

Algorithm A looks up $HT[\vec{b}|_i\|\vec{\pi}|_i\|M_i\|m_{i-1}\|X_{i-1}] = (c, y, h, h_i)$ and $GT[\pi^*, h_i] = g_i$. (We know that these entries are defined by Property 5 of a sequential forger.) If $c = b_i$ then we say that event BAD_2 occurred and A aborts. If $c \neq b_i$ then we have that $\rho^*(y) = g_i + \mu_i = \pi^*(X_i)$, so algorithm A outputs (X_i, y) as the claw for (π^*, ρ^*).

Algorithm A wins the game if forger S wins and neither of $\textsc{Bad}_1$ and $\textsc{Bad}_2$ occurs, i.e.

$$\begin{aligned}\Pr[\,\mathsf{A}\text{ wins}\,] &= \Pr\left[\,\mathsf{S}\text{ wins}\wedge\overline{\textsc{Bad}_1}\wedge\overline{\textsc{Bad}_2}\,\right]\\ &\geq \Pr\left[\,\mathsf{S}\text{ wins}\mid\overline{\textsc{Bad}_1}\wedge\overline{\textsc{Bad}_2}\,\right]\cdot\Pr\left[\,\overline{\textsc{Bad}_2}\,\right]-\Pr\left[\,\overline{\textsc{Bad}_1}\,\right]\end{aligned}$$

In the event $\overline{\textsc{Bad}_1}\wedge\overline{\textsc{Bad}_2}$ S's environment is distributed exactly as in a real attack, so

$$\Pr\left[\,\mathsf{S}\text{ wins}\mid\overline{\textsc{Bad}_1}\wedge\overline{\textsc{Bad}_2}\,\right] = \epsilon\ .$$

Event $\textsc{Bad}_1$ occurs only if either of $h_n^{(0)}=h^{(0)}\oplus h_{n-1}$ or $h_n^{(1)}=h^{(1)}\oplus h_{n-1}$ already occurred in $GT[\cdot,\cdot]$, where $h^{(0)},h^{(1)}$ are fresh random ℓ-bit strings. The probability that this happens is twice the number of entries in $GT[\cdot,\cdot]$ divided by 2^ℓ. Summing over all q_{H} queries to $\mathrm{H}(\cdot)$ we have

$$\Pr[\,\textsc{Bad}_1\,] \leq \frac{2q_{\mathrm{H}}(2q_{\mathrm{H}}+q_{\mathrm{G}})}{2^\ell}\ .$$

Since for the forgery F's view is independent of the bit c, we have that

$$\Pr\left[\,\overline{\textsc{Bad}_2}\,\right] = \frac{1}{2}\ .$$

To estimate the running time of A, we ignore all costs other than permutation evaluations. Each $\mathrm{H}(\cdot)$ query explicitly induces $2/d$ permutation evaluations on average, but hides an another evaluation to maintain the graph structure. The verification of the forgery takes up to $n_{\max}$ permutations. Overall, we have that

$$t' \leq t+((2/d+1)q_{\mathrm{H}}+n_{\max})\cdot t_\pi\ .$$

$\blacksquare$