Efficient Sequential Aggregate Signed Data

Gregory Neven

Department of Electrical Engineering
Katholieke Universiteit Leuven
Kasteelpark Arenberg 10, B-3001 Heverlee-Leuven, Belgium
Gregory.Neven@esat.kuleuven.be
http://www.neven.org

Abstract. We generalize the concept of sequential aggregate signatures (SAS), proposed by Lysyanskaya, Micali, Reyzin, and Shacham (LMRS) 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 LMRS scheme. 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.

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 roughly the same size as a single signature. 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 have to communicate over energy-consuming wireless channels.

Drawbacks of existing schemes. Only three instantiations of (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]

N. Smart (Ed.): EUROCRYPT 2008, LNCS 4965, pp. 52–69, 2008.

trapdoor permutations, but that with some tricks can be instantiated with RSA. All three schemes have some drawbacks though.

Pairings were only 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: 160 bits for $\mathcal{BGLS}$ and 320 bits for $\mathcal{LOSSW}$, versus 1024 bits for $\mathcal{LMRS}$ for a security level of 80 bits.

Finally, none of the existing schemes give the signers much freedom in choosing their own security parameters. 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 here the signers have no freedom whatsoever. A limited amount of freedom is allowed in the $\mathcal{LMRS}$ scheme, but signers have to be 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,BN07,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 $\boldsymbol{pk} = (pk_1, \ldots, pk_n)$ and messages $\boldsymbol{M} = (M_1, \ldots, M_n)$ to indicating that Σ correctly authenticates M_i under pk_i for $1 \leq i \leq n$, or $(\perp, \perp)$ 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 messages $\sum_{i=1}^{n} |M_i|$.

[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.

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 assumptions 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. 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; P for a pairing operation; E for a (multi-)exponentiation; and M for a multiplication. We give best/worst-case bounds for the overhead of the $\mathcal{SASD}$ and $\mathcal{MSD}$ schemes, as they depend on the length of the messages being signed.

Scheme	Type	KOSK	RO	Inst	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	$[\ell, k_{\mathrm{f}} + \ell]$	$1\mathrm{E} + 2n\mathrm{M}$	$2n$M
$\mathcal{SAS}$	SAS	N	Y	R,F	$k_{\mathrm{f}} + \ell$	$1\mathrm{E} + 2n\mathrm{M}$	$2n$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	$[\ell, nk_{\mathrm{f}} + \ell]$	1E	$2n$M

We then 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 like 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 proof, which requires complex "query bookkeeping" for the simulation to go through. The impact on efficiency is spectacular (see Table 1): verification takes $2n$ multiplications, signing takes one exponentiation and $2n$ multiplications, and this at a bandwidth overhead of only 160 bits, which until now was the exclusive privilege of pairing-based schemes. Moreover, the scheme allows signers to mix-and-match security parameters at will, allowing much more flexibility for use in the real world.

There is a small caveat here, namely that the promised overhead only holds if the messages being signed are of a (modest) minimum length. To show that our efficiency gains are not just due to the generalization of the primitive, we additionally present a "purebred" SAS scheme that has a typical overhead of 1184 bits, but that otherwise shares all the advantages offered by the $\mathcal{SASD}$ scheme.

Multi-signatures. A multi-signature (MS) scheme [IN83] is the natural equivalent of an (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 mainly 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, and $x\|y$ denotes a bit string from which x and y can be unambiguously reconstructed. If $k \in \mathbb{N}$, S is a set, and $y \in S$, then $\boldsymbol{x} = (x_1, \ldots, x_k) \in S^k$ is a k-dimensional vector, $\boldsymbol{x}\|y$ is the $(k+1)$-dimensional vector $(x_1, \ldots, x_k, y)$, and $\boldsymbol{x}|_i = (x_1, \ldots, x_i)$. Let ε and $\boldsymbol{\varepsilon}$ 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 a 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 $(\boldsymbol{pk}, \boldsymbol{M}) \leftarrow \mathsf{Vf}(\Sigma_n)$. This algorithm either returns lists of n public keys $\boldsymbol{pk}$ and messages $\boldsymbol{M}$, indicating that the signature correctly authenticates message M_i under public key pk_i for $1 \le i \le n$, or returns $(\perp, \perp)$ to indicate rejection. Correctness requires that the verification algorithm returns $(\boldsymbol{pk}, \boldsymbol{M})$ with probability one when the signed data is honestly generated by all signers 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 $\mathsf{Vf}(\Sigma) = (\boldsymbol{pk}, \boldsymbol{M}) \neq (\perp, \perp)$ and there exists an index $1 \leq i \leq |\boldsymbol{pk}|$ 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 $\mathsf{Vf}(\Sigma_{i-1}) = (\boldsymbol{pk}|_{i-1}, \boldsymbol{M}|_{i-1})$.

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). 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, 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 $\pi(x)$, that given ρ, x compute $\rho(x)$, and that given π^{-1}, x compute $\pi^{-1}(x)$ for any $x \in D_\pi$. Let t_π denote the time needed to compute $\pi(x)$. A claw-finding algorithm A is said to (t, ϵ)-break Π if it runs in time at most t and

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

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

Algorithm $\mathsf{Vf}^{\mathrm{H},\mathrm{G}}(\Sigma)$:
- Parse Σ as $(\boldsymbol{\pi}, m_n, X_n, h_n)$, $n = |\boldsymbol{\pi}|$
- For $i = n, \ldots, 1$ do
  - If $|\mathbb{G}_{\pi_i}| < 2^\ell$ then return $(\perp,\perp)$
  - $g_i \leftarrow \mathrm{G}_{\pi_i}(h_i)$; $\mu_i \leftarrow \pi_i(X_i) - g_i$
  - $M_i \| m_{i-1} \| X_{i-1} \leftarrow \mathsf{dec}_{\pi_i}(m_i, \mu_i)$
  - $h_{i-1} \leftarrow h_i \oplus \mathrm{H}(\boldsymbol{\pi}|_i, M_i, m_{i-1}, X_{i-1})$
- If $(m_0, X_0, h_0) = (\varepsilon, \varepsilon, 0^\ell)$
- Then return $\big(\boldsymbol{\pi}, \boldsymbol{M} = (M_1, \ldots, M_n)\big)$
- 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 describes a permutation over D_{π_i}. Indeed, unlike the LMRS scheme, the security analysis of our scheme points out that the security of an honest signer

is *not* affected by adversarially generated keys of cosigners, and thereby allows cheaper instantiations based on uncertified permutations such as low-exponent RSA and factoring. The real reason only becomes clear in the details of the security proof, but intuitively the difference is that in our scheme the data embedded in X_i, namely $M_i \| m_{i-1} \| X_{i-1}$, is passed as an extra argument to the hash function $\mathrm{H}(\cdot)$, as was done in the signature scheme with message recovery sketched above. The same trick cannot be applied to the LMRS scheme though because the embedded data (the previous signature) can only be recovered *after* evaluating the hash function. Instead, Lysyanskaya et al. overcome this problem by simulating random oracles with range $\mathbb{G}_\pi$ by choosing $x \xleftarrow{\$} D_\pi$ and returning $\pi(x)$. For the simulation to be correct, they rely on the fact that even adversarially generated π are permutations. We refer to the security proof for more details.

Also note that signers can independently choose their own value of the security parameter k; for the system-wide parameter ℓ, a comfortably high value (e.g. $\ell = 256$ or even 512) can be agreed upon 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 however (see below) the net overhead varies from ℓ bits when sufficiently long messages are being signed (in particular $|M_i| \geq k_i - k_{i-1}$), up to $\ell + \max(k_1, \ldots, k_n)$ bits for short messages.

Finally, the list of public keys $\boldsymbol{\pi}$ contained in the signed data can of course be omitted from the transmission if the verifier already knows them.

4 Instantiating Our Construction

Instantiations from RSA. An RSA key generator [RSA78] is a randomized algorithm $\mathsf{Kg}_{\mathrm{RSA}}$ that on input 1^k outputs tuples (N, e, d) where $N = pq$ is a k-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

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

One can associate a claw-free permutation family to $\mathsf{Kg}_{\mathrm{RSA}}$ by taking $\rho(x) = x^e \cdot y \bmod N$, where $y \xleftarrow{\$} \mathbb{Z}_N^*$. 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 LMRS scheme is that small verification exponents can be used, e.g. $e = 3$ or $e = 65537$, thereby reducing the cost of an exponentiation with exponent e to that of a couple of multiplications. Several options exist 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 $\mathbb{G}_{N,e} = \{0\|x \ : \ x \in \{0,1\}^{k-1}\}$ with the XOR operation. 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 time.

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 -1 is a non-square modulo N with Jacobi symbol $(-1|N) = +1$, and that 2 is a non-square with Jacobi symbol $(2|N) = -1$. Also, each square modulo N has four square roots (x_1, x_2, x_3, x_4) such that $x_1 = -x_2 \bmod N$, $x_3 = -x_4 \bmod N$, $(x_1|N) = (x_2|N) = +1$, and $(x_2|N) = (x_3|N) = -1$. Consider the permutation $\pi : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined as

$$\pi(x) \ = \ \begin{cases} x^2 \mod N \text{ if } (x|N) = +1 \text{ and } x < N/2 \\ -x^2 \mod N \text{ if } (x|N) = +1 \text{ and } x > N/2 \\ 2x^2 \mod N \text{ if } (x|N) = -1 \text{ and } x < N/2 \\ -2x^2 \mod N \text{ if } (x|N) = -1 \text{ and } x > N/2 \ . \end{cases}$$

Note that the Jacobi symbol $(x|N)$ can be computed in time $O(|N|^2)$ without knowing the factorization of N. 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 $(x_1|N) = (x_{-1}|N) = +1$, $x_1 < x_{-1}$, $(x_2|N) = (x_{-2}|N) = -1$, $x_2 < x_{-2}$. 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 $r \xleftarrow{\$} \mathbb{Z}_N^*$. Algorithm A is said to (t, ϵ)-factor $\mathsf{Kg}_{\mathrm{Wil}}$ if it runs in time at most t and

$$\Pr\left[x \in \{p, q\} \ : \ (N, p, q) \xleftarrow{\$} \mathsf{Kg}_{\mathrm{Wil}} \ ; \ x \xleftarrow{\$} \mathsf{A}(N) \right] \ \geq \ \epsilon \ .$$

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 1. *If there exists a forger* F *that* $(t, q_S, q_H, q_G, n_{\max}, \epsilon)$*-breaks* $\mathcal{SASD}$ *in the random oracle model, then there exists a claw-finding algorithm* A *that* (t', ϵ')*-breaks* Π *with*

$$\epsilon' \geq \frac{\epsilon}{e(q_S+1)} - \frac{4\big(q_H + q_G + 2n_{\max}(q_S+1)\big)^2}{2^\ell}$$
$$t' \leq t + (1/d+2)\big(q_H + 2n_{\max}(q_S+1) + n_{\max}\big)\cdot t_\pi \ .$$

We prove Theorem 1 in two steps. First, we restrict our attention to a particular class of forgers that we call *sequential* forgers, defined in Definition 1. In Lemma 1 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 2 how a sequential forger can be used to find a claw in Π. The theorem then follows directly by combining Lemma 1 and Lemma 2.

Definition 1. *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;*
2. *it only makes* $\mathrm{H}(\cdot)$ *queries of the form* $\mathrm{H}(\boldsymbol{\pi}, M_n, m_{n-1}, X_{n-1})$ *such that* $n = |\boldsymbol{\pi}| \leq n_{\max}$ *and* $|\mathbb{G}_{\pi_i}| \geq 2^\ell$ *for all* $1 \leq i \leq n$*;*
3. *for each query* $\mathrm{H}(Q_n) = \mathrm{H}(\boldsymbol{\pi}, M_n, m_{n-1}, X_{n-1})$ *there exists a unique sequence of queries* $Q_1 = (\pi_1, M_1, \varepsilon, \varepsilon), Q_2 = (\boldsymbol{\pi}|_2, M_2, m_1, X_1), \ldots, Q_{n-1} = (\boldsymbol{\pi}|_{n-1}, M_{n-1}, m_{n-2}, X_{n-2})$ *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* $\Sigma_{n-1} = (\boldsymbol{\pi}, m_{n-1}, X_{n-1}, h_{n-1})$*, meaning that* $n = |\boldsymbol{\pi}| + 1 \leq n_{\max}$ *and* $\mathsf{Vf}^{\mathrm{H},\mathrm{G}}(\Sigma_{n-1}) \neq (\perp, \perp)$*. Also, before making such a signing query, it makes a random oracle query* $\mathrm{H}(\boldsymbol{\pi}, M_{n-1}, m_{n-2}, X_{n-2})$ *and all random oracle queries needed for the verification* $\mathsf{Vf}^{\mathrm{H},\mathrm{G}}(\Sigma_{n-1})$*;*
5. *it only outputs valid forgeries* $\Sigma = (\boldsymbol{\pi}, m_n, X_n, h_n)$*, meaning that* $n = |\boldsymbol{\pi}| \leq n_{\max}$*, that* $\mathsf{Vf}^{\mathrm{H},\mathrm{G}}(\Sigma) = (\boldsymbol{\pi}, \boldsymbol{M}) \neq (\perp, \perp)$*, 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* $\mathsf{Vf}^{\mathrm{H},\mathrm{G}}(\Sigma_{i-1}) = (\boldsymbol{\pi}|_{i-1}, \boldsymbol{M}|_{i-1})$*. Also, before halting, it makes all random oracle queries needed for the verification* $\mathsf{Vf}^{\mathrm{H},\mathrm{G}}(\Sigma_{i-1})$*.*

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 1. *If there exists a forger* F *that* $(t, q_S, q_H, q_G, n_{\max}, \epsilon)$*-breaks* $\mathcal{SASD}$*, then there exists sequential forger* S *that* $(t', q_S, q'_H, q'_G, n_{\max}, \epsilon')$*-breaks* $\mathcal{SASD}$

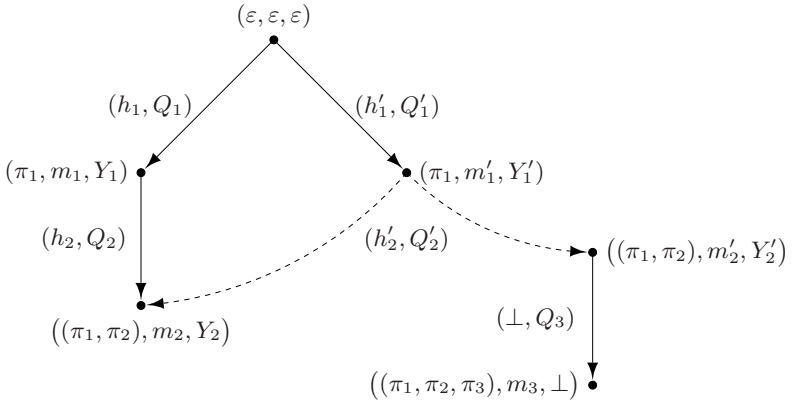

Fig. 1. The graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ maintained by algorithm S. The solid edges indicate the state of $\mathcal{G}$ after F made sequential $\mathrm{H}(\cdot)$ queries $Q_1 = (\pi_1, M_1, \varepsilon, \varepsilon)$, $Q_2 = \big((\pi_1, \pi_2), M_2, m_1, X_1\big)$, $Q_1' = (\pi_1, M_1', \varepsilon, \varepsilon)$, and a non-sequential query $Q_3 = \big((\pi_1, \pi_2, \pi_3), M_3, m_2, X_2\big)$. 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 BAD to occur.

with

$$\epsilon' \geq \epsilon - \frac{2\big(q_{\mathrm{H}} + q_{\mathrm{G}} + 2n_{\max}(q_{\mathrm{S}} + 1)\big)^2}{2^{\ell}} \tag{1}$$

$$q_{\mathrm{H}}' \leq q_{\mathrm{H}} + n_{\max}(q_{\mathrm{S}} + 1)$$

$$q_{\mathrm{G}}' \leq q_{\mathrm{H}} + q_{\mathrm{G}} + 2n_{\max}(q_{\mathrm{S}} + 1)$$

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

Proof. Given a non-sequential forger F, we build a sequential forger S as follows. Given input π^* and access to oracles $\mathrm{H}'(\cdot)$, $\mathrm{G}'_{\cdot}(\cdot)$, and $\mathsf{Sign}'(\pi^{*-1}, \cdot, \cdot)$, algorithm S 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 1, 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 will 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 Fig. 1. Each node is uniquely identified by a tuple $(\boldsymbol{\pi}, m, Y) \in \mathcal{V}$, 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 $\mathrm{H}(Q)$ 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 $\mathrm{H}'(Q)$ for queries in the sequential tree, and random values for all other queries.

When F makes a new query $\mathrm{H}(Q_n) = \mathrm{H}(\boldsymbol{\pi}, M_n, m_{n-1}, X_{n-1})$, S adds a new edge to $\mathcal{G}$ 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 $v_\mathrm{t} = (\boldsymbol{\pi}|_{n-1}, m_{n-1}, Y_{n-1})$ 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 $v_\mathrm{h} = (\boldsymbol{\pi}, m_n, Y_n)$ 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 $v_\mathrm{t} = (\boldsymbol{\pi}|_{n-1}, m_{n-1}, Y_{n-1})$ where $Y_n = \pi_{n-1}(X_{n-1})$ and head node $v_\mathrm{h} = (\boldsymbol{\pi}, m_n, \perp)$ 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 Fig. 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 Fig. 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 Bad occurs when this happens.

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

Proof. Let Σ_n be parsed as $(\boldsymbol{\pi}, m_n, X_n, h_n)$. We prove the claim by induction on the number of signatures n contained in Σ. 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 $\mathsf{Vf}^{\mathrm{H},\mathrm{G}}(\Sigma_n) = (\boldsymbol{\pi}, \boldsymbol{M})$, where $Q_i = (\boldsymbol{\pi}|_i, M_i, m_{i-1}, X_{i-1})$, and let $h'_1, \ldots, h'_{n-1}$ be the intermediate values obtained during the evaluation. If Σ_n is valid, then $\Sigma_{n-1} = (\boldsymbol{\pi}|_{n-1}, m_{n-1}, X_{n-1}, h'_{n-1})$ is also valid, namely $\mathsf{Vf}^{\mathrm{H},\mathrm{G}}(\Sigma_{n-1}) = (\boldsymbol{\pi}|_{n-1}, \boldsymbol{M}|_{n-1})$. By the induction hypothesis, F must thus have made the queries $\mathrm{H}(Q_1), \ldots, \mathrm{H}(Q_{n-1})$ sequentially, so they must be 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 $h_i = h'_i = \mathrm{H}(Q_1) \oplus \ldots \mathrm{H}(Q_i)$ for $1 \le i \le n-1$, 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 $(\bot, Q_n)$ with tail node $\big(\boldsymbol{\pi}|_{n-1}, m_{n-1}, Y_{n-1} = \pi_{n-1}(X_{n-1})\big)$ not connected to the sequential tree. When F subsequently queries $\mathrm{H}(Q_{n-1})$, S adds edge (h_{n-1}, Q_{n-1}) to the sequential tree with head node $\big(\boldsymbol{\pi}|_{n-1}, m_{n-1}, Y'_{n-1} = \mu_{n-1} + \mathrm{G}_{\pi_{n-1}}(h_{n-1})\big)$. 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 Bad to occur. So if 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 1 by first verifying Σ_{n-1} and simulating an additional query $\mathrm{H}(\boldsymbol{\pi} \| \pi^*, M_n, m_{n-1}, X_{n-1})$. 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, if the event Bad does not occur, all $\mathrm{H}(\cdot)$ queries involved in the verification $\mathsf{Vf}^{\mathrm{H},\mathrm{G}}(\Sigma_{n-1}) = (\boldsymbol{\pi}, \boldsymbol{M}) \neq (\bot, \bot)$ are sequential, so we also have that $\mathsf{Vf}^{\mathrm{H}',\mathrm{G}'}(\Sigma_{n-1}) = (\boldsymbol{\pi}, \boldsymbol{M})$.

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 $\mathsf{Sign}(\pi^{*-1}, \cdot, \cdot)$ queries hold. Again, we rely here on Claim 5 to guarantee that if event Bad does not occur, then any valid signed data under random oracles $\mathrm{H}(\cdot), \mathrm{G}.(\cdot)$ is also valid under $\mathrm{H}'(\cdot), \mathrm{G}'.(\cdot)$.

It is clear that S is successful in breaking $\mathcal{SASD}$ whenever F is and event Bad does not occur. We want to bound the probability that Bad occurs. A detailed proof of the following claim, as well as a pseudo-code description of S and precise bounds on its running time and success probability, are provided in the full version [Nev08]. An intuition is given below.

Claim. The probability that event Bad happens is at most

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

To see why the claim is true, observe that event BAD occurs when during the processing of a sequential query $\mathrm{H}(Q_n)$, the head node $\big(\boldsymbol{\pi}, m_n, Y_n = \mu_n + \mathrm{G}_{\pi_n}(h_n)\big)$ of the created edge coincides with an existing node in the graph. At this point F's view is still independent of the value of $\mathrm{H}(Q_n)$, and therefore also of $h_n = h_{n-1} \oplus \mathrm{H}(Q_n)$. The probability that F previously queried $g_n = \mathrm{G}_{\pi_n}(h_n)$ is therefore at most its number of queries to $\mathrm{G}.(\cdot)$ divided by 2^ℓ. If it did not previously make this query, then g_n is a random value from D_{π_n}, and hence so is $Y_n = \mu_n + g_n$. Because for sequential queries we insisted that $|\mathbb{G}_{\pi_n}| \geq 2^\ell$, the probability that Y_n coincides with any of the existing nodes in $\mathcal{G}$ is at most the total number of nodes in the graph divided by 2^ℓ. Summing over all $\mathrm{H}(\cdot)$ queries yields the bound mentioned above. □

The next lemma shows that any sequential forger S can be turned into a claw-finding algorithm for Π. The proof reuses ideas from [BR96,Cor00,LMRS04].

Lemma 2. *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*

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

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 $(\boldsymbol{\pi}, M_n, m_{n-1}, X_{n-1})$. 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 $h \xleftarrow{\$} \{0,1\}^\ell$, 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 $h \xleftarrow{\$} \{0,1\}^\ell$ and $c \xleftarrow{\delta} \{0,1\}$, 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 $x \xleftarrow{\$} D_{\pi^*}$ and computes $g_n \leftarrow \pi^*(x) - \mu_n$ until $g_n \in \mathbb{G}_{\pi^*}$. If $c = 1$ then A repeatedly chooses $x \xleftarrow{\$} D_{\pi^*}$ 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 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 randomly 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 $(\boldsymbol{\pi}, m_{n-1}, X_{n-1}, h_{n-1})$. Algorithm A looks up the entry $HT[\boldsymbol{\pi} \| \pi^*, M_n, m_{n-1}, X_{n-1}] = (c, X_n, h_n)$,

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 $\Sigma_n = (\boldsymbol{\pi} \| \pi^*, m_n, X_n, h_n)$. 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 $\Sigma_n = (\boldsymbol{\pi}, m_n, X_n, h_n)$. By Property 5 the forgery is valid, so $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_n) = (\boldsymbol{\pi}, \boldsymbol{M})$ 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 $\mathsf{Vf}^{\mathrm{H,G}}(\Sigma_{i-1}) = (\boldsymbol{\pi}|_{i-1}, \boldsymbol{M}|_{i-1})$. 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 $HT[\boldsymbol{\pi}|_i, M_i, m_{i-1}, X_{i-1}] = (c, y, h_i)$ 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^*).

A detailed analysis in support of the bounds stated in the lemma reuses techniques due to Coron [Cor00], and is given in the full version [Nev08]. □

We can now shed some more technical light on how our security proof avoids relying on the family of permutations being certified like the LMRS scheme does. The LMRS scheme uses a full-domain random oracle, much like our $\mathrm{G}.(\cdot)$ oracle. 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. The usual trick of generating a random preimage $x \xleftarrow{\$} D_\pi$ and computing the output as $\pi(x)$ only gives the correct distribution if π is a permutation, hence the requirement that Π be certified. In our proof, the algorithm A only needs to know preimages related to queries $\mathrm{G}_{\pi^*}(\cdot)$, but not for queries $\mathrm{G}_\pi(\cdot)$ with $\pi \neq \pi^*$. It can therefore sample random elements from $\mathbb{G}_\pi$ directly.

6 Variations on the Main Construction

Sequential aggregate signatures. If for some reason the message recovery functionality is undesirable, then the following "purebred" sequential aggregate signature scheme $\mathcal{SAS}$ is easily derived from our $\mathcal{SASD}$ scheme. 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 again depends on various issues such as the encoding efficiency and the permutation being used, but for signers with security parameters $k_1, \ldots, k_n$ will typically be about $\max(k_1, \ldots, k_n) + \ell$ bits.

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 $\mathsf{Sign}^{\mathrm{H},\mathrm{G}}(\pi^{-1}, M_n, \boldsymbol{\pi}, \boldsymbol{M}, \sigma_{n-1})$:
If $n = |\boldsymbol{\pi}| = 1$ then $\sigma_0 \leftarrow (\varepsilon, \varepsilon, 0^\ell)$
Parse σ_{n-1} as $(x_{n-1}, X_{n-1}, h_{n-1})$
If $\mathsf{Vf}^{\mathrm{H},\mathrm{G}}(\boldsymbol{\pi}, \boldsymbol{M}, \sigma_{n-1}) = 0$ then return $\perp$
$(x_n, \xi_n) \leftarrow \mathsf{enc}_{\pi_n}(x_{n-1} \| X_{n-1})$
$h_n \leftarrow h_{n-1} \oplus \mathrm{H}(\boldsymbol{\pi} \| \pi_n, \boldsymbol{M} \| M_n, x_{n-1}, X_{n-1})$
$g_n \leftarrow \mathrm{G}_{\pi_n}(h_n)$
$X_n \leftarrow \pi_n^{-1}(g_n + \xi_n)$
Return $\sigma_n \leftarrow (x_n, X_n, h_n)$

Algorithm $\mathsf{Vf}^{\mathrm{H},\mathrm{G}}(\boldsymbol{\pi}, \boldsymbol{M}, \sigma_n)$:
Parse σ_n as (x_n, X_n, h_n), $n = |\boldsymbol{\pi}|$
For $i = n, \ldots, 1$ do
  If $|\mathbb{G}_{\pi_i}| < 2^\ell$ then return 0
  $g_i \leftarrow \mathrm{G}_{\pi_i}(h_i)$; $\xi_i \leftarrow \pi_i(X_i) - g_i$
  $x_{i-1} \| X_{i-1} \leftarrow \mathsf{dec}_{\pi_i}(m_i, \mu_i)$
  $h_{i-1} \leftarrow h_i \oplus \mathrm{H}(\boldsymbol{\pi}|_i, \boldsymbol{M}|_i, x_{i-1}, X_{i-1})$
If $(x_0, X_0, h_0) = (\varepsilon, \varepsilon, 0^\ell)$ then return 1
Else return 0.

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

Achieving tight security. Closer inspection of Theorem 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 LMRS scheme in [BNN07].) We refer to the full version [Nev08] for details.

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 $\boldsymbol{\sigma}$ into the final signed data via $\Sigma \xleftarrow{\$} \mathsf{Comb}(\boldsymbol{pk}, M, \boldsymbol{\sigma})$. The verification algorithm $\mathsf{Vf}(\Sigma)$ returns $(\boldsymbol{pk}, M)$ to indicate that Σ is valid for signers $\boldsymbol{pk}$ and message M, or returns $(\perp, \perp)$ to indicate rejection. Correctness requires that $\mathsf{Vf}(\Sigma) = (pk, M)$ if all signers behave honestly.

In the experiment defining security, the forger F is given a freshly generated pk^* as input, and has access to a signing oracle $\mathsf{Sign}(sk^*, \cdot)$. It wins if it outputs a forgery Σ such that $\mathsf{Vf}(\Sigma) = (\boldsymbol{pk}, M) \neq (\perp, \perp)$ with $pk_i = pk^*$ for some $1 \leq i \leq |\boldsymbol{pk}|$ 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 $\mathrm{H} : \{0,1\}^* \to \{0,1\}^\ell$ and $\mathrm{G}_\pi : \{0,1\}^\ell \to \mathbb{G}_\pi$ be random oracles, exactly as for the $\mathcal{SASD}$ scheme. The encoding and decoding functions are different though: we assume that $\mathsf{enc}_\pi : \{0,1\}^* \to \mathbb{G}_\pi$, $\mathsf{enc}_{(\pi_1,\ldots,\pi_n)} : \{0,1\}^* \to \{0,1\}^*$, and $\mathsf{dec}_{(\pi_1,\ldots,\pi_n)} : \{0,1\}^* \times \mathbb{G}_{\pi_1} \times \ldots \times \mathbb{G}_{\pi_n}$ are such that $\mathsf{enc}_{\pi_i}(M)$ outputs a group element $\mu_i \in \mathbb{G}_{\pi_i}$; $\mathsf{enc}_{\boldsymbol{\pi}}(M)$ outputs a partial message m; and the injective function $\mathsf{dec}_{\boldsymbol{\pi}}(m, \boldsymbol{\mu})$ reconstructs the original message M. Key generation is consists again of generating a random permutation π as public key and its inverse π^{-1} as secret key; the other algorithms are described below.

Algorithm $\mathsf{Sign}^{\mathrm{H},\mathrm{G}}(\pi^{-1}, M)$:
$\mu \leftarrow \mathsf{enc}_\pi(M)$; $h \leftarrow \mathrm{H}(M)$
$g \leftarrow \mathrm{G}_\pi(h)$; $X \leftarrow \pi^{-1}(\mu + g)$
Return X

Algorithm $\mathsf{Comb}^{\mathrm{H},\mathrm{G}}(\boldsymbol{\pi}, M, \boldsymbol{X})$:
$m \leftarrow \mathsf{enc}_{\boldsymbol{\pi}}(M)$; $h \leftarrow \mathrm{H}(M)$
Return $\Sigma \leftarrow (\boldsymbol{\pi}, m, \boldsymbol{X}, h)$

Algorithm $\mathsf{Vf}^{\mathrm{H},\mathrm{G}}(\Sigma)$:
Parse Σ as $(\boldsymbol{\pi}, m, \boldsymbol{X}, h)$; $n \leftarrow |\boldsymbol{\pi}|$
If $|\boldsymbol{X}| \neq n$ then return $(\bot, \bot)$
For $i = 1, \ldots, n$ do
  $g_i \leftarrow \mathrm{G}_{\pi_i}(h_i)$; $\mu_i \leftarrow \pi_i(X_i) - g_i$
$M \leftarrow \mathsf{dec}_{\boldsymbol{\pi}}\big(m, (\mu_1, \ldots, \mu_n)\big)$
If $\mathrm{H}(M) = h$ then return $(\boldsymbol{\pi}, M)$
Else return $(\bot, \bot)$.

Security. The following theorem states that the $\mathcal{MSD}$ scheme is secure if the permutation family is claw-free. The proof uses techniques from [BR96,Cor00] and is provided in the full version [Nev08].

Theorem 2. *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*

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

$$t' \leq t + \frac{q_{\mathrm{H}} + q_{\mathrm{S}} + n_{\max} + 1}{d} \cdot t_\pi \ .$$

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 $\mathsf{enc}_{\boldsymbol{\pi}}(M)$ 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 $\boldsymbol{\pi}$ in the computation of $h \leftarrow \mathrm{H}(\boldsymbol{\pi}, M)$, because otherwise there may exist (contrived) encoding algorithms that make the scheme insecure. Details are left as an exercise to the reader.

Acknowledgements

We thank the anonymous referees of EUROCRYPT 2008 for their helpful suggestions. The author is a Postdoctoral Fellow of the Research Foundation – Flanders (FWO-Vlaanderen). This work was supported in part by the European Commission through the IST Program under Contract IST-2002-507932 ECRYPT, and in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

References

BFM88. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications. In: 20th ACM STOC, pp. 103–112. ACM Press, New York (1988)

BGLS03. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003)

BGOY07. Boldyreva, A., Gentry, C., O'Neill, A., Yum, D.H.: Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In: ACM CCS 2007, pp. 276–285. ACM Press, New York (2007)

BN06. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: ACM CCS 2006, pp. 390–399. ACM Press, New York (2006)

BN07. Bellare, M., Neven, G.: Identity-based multi-signatures from RSA. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 145–162. Springer, Heidelberg (2006)

BNN07. Bellare, M., Namprempre, C., Neven, G.: Unrestricted aggregate signatures. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 411–422. Springer, Heidelberg (2007)

Bol03. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 31–46. Springer, Heidelberg (2002)

BR93. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: ACM CCS 1993, pp. 62–73. ACM Press, New York (1993)

BR96. Bellare, M., Rogaway, P.: The exact security of digital signatures: How to sign with RSA and Rabin. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (1996)

BY96. Bellare, M., Yung, M.: Certifying permutations: Noninteractive zero-knowledge based on any trapdoor permutation. Journal of Cryptology 9(3), 149–166 (1996)

CM99. Camenisch, J., Michels, M.: Proving in zero-knowledge that a number is the product of two safe primes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 107–122. Springer, Heidelberg (1999)

Cor00. Coron, J.-S.: On the exact security of full domain hash. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 229–235. Springer, Heidelberg (2000)

CPP07. Catalano, D., Pointcheval, D., Pornin, T.: Trapdoor hard-to-invert group isomorphisms and their application to password-based authentication. Journal of Cryptology 20(1), 115–149 (2007)

GR06. Gentry, C., Ramzan, Z.: Identity-based aggregate signatures. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 257–273. Springer, Heidelberg (2006)

HOT04. Hayashi, R., Okamoto, T., Tanaka, K.: An RSA family of trap-door permutations with a common domain and its applications. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 291–304. Springer, Heidelberg (2004)

IN83. Itakura, K., Nakamura, K.: A public-key cryptosystem suitable for digital multisignatures. NEC Research & Development 71, 1–8 (1983)

KLS00. Kent, S., Lynn, C., Seo, K.: Secure border gateway protocol (S-BGP). IEEE JSAC 18(4), 582–592 (2000)

KW03. Katz, J., Wang, N.: Efficiency improvements for signature schemes with tight security reductions. In: ACM CCS 2003, pp. 155–164. ACM Press, New York (2003)

LMRS04. Lysyanskaya, A., Micali, S., Reyzin, L., Shacham, H.: Sequential aggregate signatures from trapdoor permutations. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 74–90. Springer, Heidelberg (2004)

LOS+06. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 465–485. Springer, Heidelberg (2006)

Nev08. Neven, G.: Efficient sequential aggregate signed data. Cryptology ePrint Archive (2008)

RSA78. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signature and public-key cryptosystems. Communications of the Association for Computing Machinery 21(2), 120–126 (1978)