How to Aggregate the CL Signature Scheme

Dominique Schröder*

University of Maryland, USA
schroeder@me.com
www.dominique-schroeder.de

Abstract. We present an aggregate signature scheme whose public key consists of only two group elements. It is therefore the first sequential aggregate signature scheme with short keys in the standard model. Our construction relies on the Camenisch-Lysyanskaya signature scheme (Crypto 2004) and is provably secure under the LRSW assumption. Moreover, we develop a novel aggregation technique that we call *aggregate-extension technique*. The basic idea is to extend the aggregate by a single element and to use this additional space to "store" some information that would be lost due to the compression of the signatures. We believe that this technique might be of independent interest.

1 Introduction

Aggregate signature schemes allow the combination of several signatures into a single element, the aggregate, that has roughly the same size as an ordinary signature. Here, we consider the sequential case where a signer receives an aggregate-so-far, adds its own signature to the aggregate and forwards the aggregate (containing the new signature) to the next signer. The size of the aggregate is independent of the number of signers, i.e., it has the roughly the same size as an ordinary signature scheme. Typical applications for such schemes are sensor networks where communication is prohibitively expensive [2]. Since the transmission range of each sensor is limited, the sensor forwards its data to the next sensor node towards the base station. Moreover, each sensor signs its measurement to prevent attackers from raising a false alarm. One example of such a monitoring network is the Tsunami early warning system that is already in operation in the Indian Ocean [23]. Further applications are the compression of certificate chains [10] and secure routing protocols, such as the Secure Border Gateway Protocol (S-BGP) [6].

Public-Key Size. Efficiency refers to three kinds of costs: computational, storing data, and the cost of communication. In practice, however, computational costs play a minor role due to the rapid growth of computational power over the last decades. On the other hand, the costs of transmitting data and of storing data are essential in practice. Consequently, an entire branch of research in

* Supported in part by a DAAD postdoctoral fellowship.

V. Atluri and C. Diaz (Eds.): ESORICS 2011, LNCS 6879, pp. 298–314, 2011.

cryptography focuses on minimizing the size of transmitted data, such as short signatures, e.g., [11,14,13,8,22] and short group signatures, e.g., [9,14].

For aggregate signature schemes the size of the public key is an important measurement. The reason is that in most of the applications the public-keys are transported with the aggregate and bandwidth is expensive. Neven already pointed out that optimizing the transmitted data means reducing the size of all three elements, rather than only considering the size of the aggregate σ [31]. The size of the data transmitted from user to user increases thus linearly in the number of users; as it is difficult to reduce the message size (which is often fixed — consider for example the measurements of the water level of the Tsunami early warning system), and because the aggregate σ is constant, reducing the key size is the only way to reduce the size of the transmitted data. Additionally, Lu et al. mention in their paper that "large keys negates any benefit from reducing the signature size in a certificate chain, since the keys must be included in the certificate" [27]. In this paper, we present the first sequential aggregate signature scheme with short keys in the standard model.

As an example, we consider the size of an aggregate generated by 20 signers; a comparison of the size for the different schemes is given in the last column of Table 1. For the calculation of the values we assume that the size of each element on an elliptic curve has 160 bits and that the messages have 160 bits as well. In the case of RSA group elements we assume that the moduls has 1024 bits. Thus, in the case of BGLS we transmit $(20 + 20 + 1) * 160 = 6560$ bits. Note, that our construction is not as efficient as the LOSSW sequential aggregate signature scheme from a computational point of view[1], but it reduces the key-size — and thus the size of the transmitted data.

Our Approach. We show how to aggregate the Camenisch-Lysyanskaya signature scheme. The public key of their scheme has only two group elements and it does not rely on the random oracle heuristic. We briefly recall the CL signature scheme to explain why aggregation of this scheme is not straightforwardly possible. A signature $\sigma = (a, b, c)$ on a message M consists of three elements $g^r, g^{r\,y}, g^{r\,(x+Mxy)}$, where r is a random element, and the values x and y are stored in the private key. Unfortunately, common aggregation techniques fail in this case. If we try to aggregate the signature by simply multiplying different signatures together, then we end up with different signatures having a different randomness. One solution would be to rely on a global counter such that all signers share the same randomness. It is well known, however, that global counters are always difficult to realize. Instead, we pick up an idea of Lu et al. [27] and let the $(i+1)$th signer "re-use" the randomness r_i of the ith signer. That is, it treats the element $a_i = g^{r_i}$ as its own randomness and computes $b_{i+1} \leftarrow a_i^{y_{i+1}}$ and $c_{i+1} \leftarrow a_i^{x_{i+1}+M_{i+1}x_{i+1}y_{i+1}}$. It is easy to see that the tuple (a_i, b_{i+1}, c_{i+1}) forms a valid CL signature.

[1] Chatterjee, Hankerson, Knapp, and Menezes have recently compared the efficiency of the BGLS aggregate signature scheme with the LOSSW aggregate signature scheme [15]. The comparison shows that evaluating n pairings is not as expensive as one might have expected.

Table 1. Comparison of aggregate signature schemes

Scheme	ROM	KOSK	Size	SK/PK	Signing	Verification	Trans-Data 20 Users
BGLS [10]	YES	NO	1	1/1	1E	nP	6560 bits
LMRS-1 [28]	YES	NO	1	1/1	nE	$2n$E	24,704 bits
LMRS-2 [28]	YES	NO	1	1/1	nM	$4n$M	24,704 bits
Neven [31]	YES	NO	1	1/1	1E + $2\,n$M	$2\,n$M	24,704 bits
LOSSW [27]	NO	YES	2	162/162	2P + $n\ell$M	2P + $n\ell$M	∼63KB
Section 3.4	NO	YES	4	**2/2**	nP + $2n$E	nP+nE	9760 bits

We use the following notation: ROM means that the security proof is given in the random oracle model and "KOSK" indicates that the security is proven in the certified-key model, where the adversary has to proof knowledge about the private keys. Sizes of the aggregate or of the keys are given as the number of objects (group elements, ring elements). Note that the size of an RSA group elements is roughly 10 times the size of an elliptic curve element. n denotes the number of participants, P a pairing computation, E a (multi)-exponentiation, M for a multiplication and ℓ for the output length of a collision resistant hash function. These values, except for the last row, are a verbatim copy of [31].

Now, multiplying the tuples (a_i, b_i, c_i) and (a_i, b_{i+1}, c_{i+1}) component wise together in order to aggregate these signatures is still not sufficient. We illustrate the problem on the following toy examples and suggest a new aggregation technique that we call *aggregate-extension technique*. Let g^a, g^b be public keys and let g^{s_a} (resp. g^{s_b}) denote the signatures under the keys g^a (resp. under the key g^b). Now, think about a verification equation that computes a non-generate, bilinear map $\mathbf{e}(g^a\, g^b, g^{s_a} g^{s_b})$ to verify both signatures. The upcoming problem results from the properties of the bilinear map: $\mathbf{e}(g^a\, g^b, g^{s_a} g^{s_b}) = \mathbf{e}(g^a, g^{s_a} g^{s_b}) \cdot \mathbf{e}(g^b, g^{s_a} g^{s_b}) = \mathbf{e}(g^a, g^{s_a}) \cdot \mathbf{e}(g^a, g^{s_b}) \cdot \mathbf{e}(g^b, g^{s_a}) \cdot \mathbf{e}(g^b, g^{s_b})$. If we consider the pairs $\mathbf{e}(g^a, g^{s_b})$ and $\mathbf{e}(g^b, g^{s_a})$, then we have two signatures that must verify under the wrong keys, which is impossible. To handle these elements, we slightly extend the aggregate by a single group element D. This element serves as a "programmable memory" which stores these (undesired) elements. In the example the "memory" element D would have the form $g^{as_b + bs_a}$ and the corresponding equation would have the form $\mathbf{e}(g^a g^b, g^{s_a} g^{s_b}) \cdot \mathbf{e}(g, g^{as_b + bs_a})^{-1}$. Finally, we apply the aggregate extension technique to the CL signature scheme.

Knowledge-of-Secret-Key. As shown in Figure 1, our construction is secure in the knowledge-of-secret-key setting (KOSK), where the adversary must prove knowledge of each private key corresponding to any maliciously generated public key. This setting can be implemented through a CA that asks the user to perform a proof-of-knowledge of the secret key when registering a public key. Alternatively, all user can generate their keys jointly [30], or the public keys come with an extractable non-interactive proof of knowledge. While the construction of Lu et al. [27] relies also on this assumption, it is clear that a solution outside the KOSK is desirable. We refer the reader to [5,3,34] for a comprehensive discussion about this setting.

Related Work. Boneh et al. [10] introduced the concept of aggregate signatures and gave a first instantiation that is provably secure in the random oracle model. At Eurocrypt 2004, Lysyanskaya et al. suggested the concept of sequential aggregate signatures, proposed a formal security model, and gave a first solution based on general assumptions [28] that is also secure in the random oracle. The first instantiation that is secure in the standard model, but in a model that is weaker than one of [28], was proposed by Lu et al. [27]. Neven suggested at Eurocrypt 2008 the notion of sequential aggregate signed data, which generalized sequential aggregate signature in the sense that these scheme compress the whole data [31]. Fischlin et al. adopt the notion of history-freeness from Eikemeyer et al. [18] to the case of sequential aggregate signatures [19]. The basic idea is to allow the aggregate-so-far only depend (explicitly) on the local message and signing key, but not on the previous messages and public keys in the sequence. The benefit of this notion is that one does not need to take care about the key management and that expensive verification queries are not necessary anymore. Eikemeyer et al. considered the notion of history-freeness only in the context of aggregate message authentication codes [25]. Multisignatures are similar in the sense that a group of signers sign the same message [24]. Many researches suggested different solutions, such as, e.g., [21,33,32]. However, the size of the multisignature proposed by some solutions grows linear in the number of signers [24] and some such schemes cannot be considered as secure [21]. Boldyreva simplified and generalized the security model of [30]. The author also gave the first solution that does not require a priori knowledge of a subgroup of signers and that is provably secure in the random oracle [5]. The first solution that is secure in the standard model has been suggested by Lu et al. [27]. Recently, Boldyreva et al. introduced the concept of ordered multisignatures [7], where the signers attest to a common message as well as to the order in which they signed.

2 Preliminaries

Notations. If x is a string then $|x|$ denotes its length, and if S is a set $|S|$ denotes its size. By $a_1 \| \ldots \| a_n$ we denote a string encoding of $a_1, \ldots, a_n$ from which $a_1, \ldots, a_n$ are uniquely recoverable. If A is an algorithm then $y \leftarrow A(x_1, x_2, \ldots)$ denotes the operation of running A on inputs $x_1, x_2, \ldots$ and letting y denote the output of A. Unless otherwise indicated, we assume that all algorithms run in probabilistic polynomial-time and refer to them as being efficient.

2.1 Bilinear Groups

We denote by $\mathbb{G}$ and $\mathbb{G}_T$ two multiplicative groups of prime order p and consider $g \in \mathbb{G}$ such that: all group operations can be computed efficiently; g is a generator of $\mathbb{G}$; $\mathbf{e}$ is a bilinear pairing: $\mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, i.e., $\mathbf{e}$ is an efficiently computable map satisfying the following properties:

- Non-degeneracy: $\mathbf{e}(g, g) \neq 1$ and is thus a generator of $\mathbb{G}_T$;
- Bilinearity: $\forall u, v \in \mathbb{G}, \forall a, b \in \mathbb{Z}$: $\mathbf{e}\left(u^a, v^b\right) = \mathbf{e}(u, v)^{ab}$.

As a result of the bilinearity, it holds that $\mathbf{e}(g^a, g) = \mathbf{e}(g, g)^a = \mathbf{e}(g, g^a)$.

Definition 1 (Bilinear-Group Generation). *The algorithm $\mathcal{G}$ that outputs (descriptions of) $p, \mathbb{G}, \mathbb{G}_T, \mathbf{e}$ as above is called* bilinear-group generation algorithm*, and $\mathbb{G}$ is a bilinear group.*

2.2 Signature Scheme

Definition 2 (Signature Scheme). *A signature scheme* $\mathsf{DS} = (\mathsf{PKg}, \mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$ *is a tuple of algorithms:*

Parameter Generation. $\mathsf{PKg}(1^\lambda)$ *returns some global information I.*
Key Generation. $\mathsf{Kg}(I)$ *outputs a keypair (sk, pk).*
Signing. *The input of the signing algorithm* $\mathsf{Sig}(sk, M)$ *is a signing key sk and a message M; it outputs a signature σ.*
Verification. $\mathsf{Vf}(pk, M, \sigma)$ *outputs 1 iff σ is a signature on M under pk.*

The security of signature schemes is proven against existential forgery under adaptive chosen message attacks (EU-CMA) due to Goldwasser, Micali, and Rivest [20]. In this model, an adversary adaptively invokes a signing oracle and is successful if it outputs a signature on a *fresh* message.

Definition 3 (Unforgeability). *A signature scheme* DS *is* unforgeable under adaptive chosen message attacks *(EU-CMA) if for any efficient algorithm $\mathcal{A}$ the probability that the experiment* $\mathsf{Forge}_{\mathcal{A}}^{\mathsf{DS}}$ *evaluates to 1 is negligible (as a function of λ), where*

Experiment $\mathsf{Forge}_{\mathcal{A}}^{\mathsf{DS}}(\lambda)$
$I \leftarrow \mathsf{PKg}(1^\lambda)$
$(sk, pk) \leftarrow \mathsf{Kg}(I)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sig}(sk,\cdot)}(pk)$
Return 1 *iff* $\mathsf{Vf}(pk, m^*, \sigma^*) = 1$ *and $\mathcal{A}$ has never queried* $\mathsf{Sig}(sk, \cdot)$ *about m^*.*

A signature scheme DS *is* (t, q_S, ϵ)-secure *if no adversary running in time at most t, invoking the signing oracle at most q_S times, outputs a valid forgery (m^*, σ^*) with probability larger than ϵ.*

2.3 The CL Signature Scheme

The signature scheme due to Camenisch and Lysyanskaya has been introduced at CRYPTO 2004 and its security relies on the interactive LRSW assumption [14]. The LRSW assumption, due to Lysyanskaya, Rivest, Sahai, and Wolf, is hard in the generic group model (as defined by Shoup [35]) [29]. Moreover, it is widely established and it is the basis for many constructions such as, e.g., [14,1,4,12,16,26].

Assumption (LRSW Assumption). Suppose that $\mathbb{G}$ is group of prime order p and that g is a generator of $\mathbb{G}$. Let $X, Y \in \mathbb{G}$ such that $X = g^x$ and $Y = g^y$

for some $x, y \in \mathbb{Z}_p$ and let $\rho := (p, \mathbb{G}, \mathbb{G}_T, g, \mathbf{e}, X, Y)$ and let $O_{X,Y}$ be an oracle that on input a value $M \in \mathbb{Z}_p$ outputs a triplet $(a, a^y, a^{(x+Mxy)})$ for a randomly chosen $a \in \mathbb{G}$. Then for all efficient algorithms $\mathcal{A}^{O_{x,y}}$, $\nu(\lambda)$ defined as follows is a negligible function (in λ):

$$\text{Prob}[x \leftarrow \mathbb{Z}_p\,;\, y \leftarrow \mathbb{Z}_p\,;\, X \leftarrow g^x\,;\, Y \leftarrow g^y;$$
$$(Q, M, a, b, c) \leftarrow \mathcal{A}^{O_{x,y}}(\rho) : M \notin Q \wedge a \in \mathbb{G} \wedge b = a^y \wedge c = a^{x+Mxy}] = \nu(\lambda),$$

where Q is the set of oracle queries. Based on this assumption, the signature scheme consists of the following algorithms:

Parameter Generation. $\mathsf{PKg}(1^\lambda)$ executes the bilinear-group generation algorithm $\mathcal{G}$ to obtain output $(p, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, it chooses generator $g \in \mathbb{G}$ and returns $I = (p, \mathbb{G}, \mathbb{G}_T, \mathbf{e}, g)$.

Key Generation. $\mathsf{Kg}(I)$ picks two random elements $x \leftarrow \mathbb{Z}_p$ and $y \leftarrow \mathbb{Z}_p$. It sets $X \leftarrow g^x$ and $Y \leftarrow g^y$. The private key is $sk = (I, x, y)$ and the public key is $pk = (I, X, Y)$.

Signing. The input of the algorithm $\mathsf{Sig}(sk, M)$ is a secret key $sk = (I, x, y)$ and a message $M \in \mathbb{Z}_p$. It selects a random element $r \leftarrow \mathbb{Z}_p$ and computes the signature $\sigma = (a, b, c) \leftarrow (g^r, g^{r\,y}, g^{r\,(x+Mxy)})$.

Verification. To check that $\sigma = (a, b, c)$ is a valid signature on message $M \in \mathbb{Z}_p$ under a public-key $pk = (I, X, Y)$, the algorithm $\mathsf{Vf}(pk, M, \sigma)$ verifies that

$$\mathbf{e}(a, Y) = \mathbf{e}(g, b) \quad \text{and} \quad \mathbf{e}(X, a) \cdot \mathbf{e}(X, b)^M = \mathbf{e}(g, c) \quad \text{hold.}$$

The CL-signature scheme is provably secure in the standard model, assuming that the LRSW assumption holds.

Proposition 1 ([14]). *If the LRSW assumption is hard relative to $\mathcal{G}$, then the CL-signature scheme is unforgeable under adaptively chosen message attacks.*

3 Sequential Aggregate Signature

Boneh et al. proposed a new signature primitive called an aggregate signature [10]. Aggregate signature generalize multisignatures [24] as they combine several signatures, on distinct messages from different users, into a single signature that has *roughly the same size* as an ordinary signature. In such a scheme, the individual users generate their signature independently in advance. The signatures are then combined into a single aggregate by a third, maybe untrusted, party.

Sequential aggregate signature schemes (SAS) are different in the sense that aggregation is performed sequentially. Each individual signer gets as input an aggregate-so-far σ' and "adds" its own signature onto the aggregate. Signing and aggregation are therefore a combined process. The resulting aggregate has roughly the same size as a standard signature.

More formally, the input of the aggregate-signing algorithm is a secret key sk, a message M_i to be signed and an aggregate-so-far tuple $(\sigma', \mathbf{M}, \mathbf{pk})$. This tuple

consists of an aggregate σ', of a sequence of messages $\mathbf{M} = (M_1, \ldots, M_{i-1})$, and of a sequence of public keys $\mathbf{pk} = (pk_1, \ldots, pk_{i-1})$. This algorithm outputs a new aggregate σ for message and public key sequences $\mathbf{M}||M := (M_1, \ldots, M_{i-1}, M_i)$ and $\mathbf{pk}||pk := (pk_1, \ldots, pk_{i-1}, pk_i)$, such that the aggregate has roughly the same size as an ordinary signature.

3.1 Definition

Definition 4 (Sequential Aggregate Signature). *A sequential aggregate signature scheme is a tupple of efficient algorithms* $\mathsf{SAS} = (\mathsf{SeqPKg}, \mathsf{SeqKg}, \mathsf{SeqAgg}, \mathsf{SeqAggVf})$*, where:*

System Parameter. $\mathsf{SeqPKg}(1^\lambda)$ *returns the system parameters* I*.*
Key Generation. $\mathsf{SeqKg}(I)$ *generates a key pair* (sk, pk)*.*
Signature Aggregation. *The input of the aggregation algorithm* SeqAgg *is a tuple* $(sk, M_i, \sigma', \mathbf{M}, \mathbf{pk})$ *consisting of a secret key* sk*, a message* $M_i \in \{0,1\}^*$*, an aggregate* σ'*, and sequences* $\mathbf{M} = (M_1, \ldots, M_{i-1})$ *of messages and* $\mathbf{pk} = (pk_1, \ldots, pk_{i-1})$ *of public keys. It computes the aggregate* σ *for the message sequence* $\mathbf{M}||M = (M_1, \ldots, M_{i-1}, M_i)$ *and the key sequence* $\mathbf{pk}||pk = (pk_1, \ldots, pk_{i-1}, pk_i)$*.*
Aggregate Verification. *The algorithm* $\mathsf{SeqAggVf}(\sigma, \mathbf{M}, \mathbf{pk})$ *takes as input an aggregate* σ*, a sequence of messages* $\mathbf{M} = (M_1, \ldots, M_\ell)$*, as well as a sequence of public keys* $\mathbf{pk} = (pk_1, \ldots, pk_\ell)$*. It returns a bit.*

The sequential aggregate signature scheme is $\mathsf{complete}$ *if for any finite sequence of key pairs* $(sk, pk), (sk_1, pk_1), \ldots, (sk_n, pk_n) \leftarrow \mathsf{SeqKg}(1^\lambda)$*, for any (finite) sequence* $M_1, \ldots, M_n$ *of messages with* $M_i \in \{0,1\}^*$*, for any* $\sigma \leftarrow \mathsf{SeqAgg}(sk, M, \sigma', \mathbf{M}, \mathbf{pk})$ *with* $\mathsf{SeqAggVf}(\sigma', \mathbf{M}, \mathbf{pk}) = 1$ *or* $\sigma' = \emptyset$*, we have* $\mathsf{SeqAggVf}(\sigma, \mathbf{M}\|M, \mathbf{pk}\|pk) = 1$*.*

3.2 Security Model

The security of a sequential aggregate signature is defined in the chosen-key model. Thereby, the adversary gets as input a single key (referred to as the challenge key pk_c), it is allowed to choose all other keys, and has access to a sequential aggregate signing oracle $\mathsf{OSeqAgg}$ and to a key registration oracle RegKey. The input of the oracle $\mathsf{OSeqAgg}$ is a message M to be signed under the challenge key sk_c, an aggregate-so-far σ', on a set of messages $\mathbf{M}$, under public key sequence $\mathbf{pk}$; the oracle then outputs a new aggregate σ that contains also the signature σ' on the challenge message M. The second oracle RegKey takes as input a private key sk and the corresponding public key pk. The task for the adversary is to output a sequential aggregate signature such that the aggregate contains a signature on a "fresh" message M_c for the challenge key pk_c. A message is "fresh" in the usual sense, namely that it has not been sent to the aggregate signing oracle [27]. Note that the requirement that the challenge public-key pk_c has not been registered, i.e., $pk_c \notin C$, can be dropped in the KOSK because the adversary would have to compute the corresponding private key sk_c to ask such a question. Here, however, we keep this restriction to simplify the proof.

Definition 5 (Aggregate-Unforgeable). *A sequential aggregate signature scheme* $\mathsf{SAS} = (\mathsf{SeqPKg}, \mathsf{SeqKg}, \mathsf{SeqAgg}, \mathsf{SeqAggVf})$ *is* aggregate-unforgeable *if for any efficient algorithm* $\mathcal{A}$ *the probability that the experiment* $\mathsf{SeqForge}^{\mathsf{SAS}}_{\mathcal{A}}$ *evaluates to* 1 *is negligible (as a function of* λ*), where*

Experiment $\mathsf{SeqForge}^{\mathsf{SAS}}_{\mathcal{A}}(\lambda)$
$I \leftarrow \mathsf{SeqPKg}(1^\lambda)$
$(sk_c, pk_c) \leftarrow \mathsf{SeqKg}(I)$
$(\mathbf{pk}^*, \mathbf{M}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{RegKey}(\cdot,\cdot),\mathsf{OSeqAgg}(sk_c,\cdots)}(pk_c)$
Let C *denote the list of all certified key* $(sk_1, pk_1), \ldots, (sk_\ell, pk_\ell)$ *and let* M_c *be the message that corresponds to* pk_c.
Return 1 *iff* $pk_i \neq pk_j$ *for all* $i \neq j$ *and* $\mathsf{SeqAggVf}(\mathbf{pk}^*, \mathbf{M}^*, \sigma^*) = 1$ *and* $pk_c \notin C$ *and* $\mathcal{A}$ *has never sent* M_c *to* $\mathsf{OSeqAgg}(sk_c, \cdots)$.

A sequential aggregate signature scheme DS *is* (t, q_S, q_N, ϵ)-aggregate unforgeable *if no adversary running in time at most* t*, invoking the signing oracle at most* q_S *times, registering at most* q_N *keys, outputs a valid forgery* $(\mathbf{pk}^*, \mathbf{M}^*, \sigma^*)$ *with probability larger than* ϵ.

3.3 Intuition for the Construction

We consider signers who have unique key-pairs as in our signature scheme, consisting of two private values x, y, and two public keys $X = g^x, Y = g^y$ for a shared generator g of a group $\mathbb{G}$. Thereby the signature consists of three group elements A, B, and C. We view the element $A = g^r$ as the randomness; $B = g^{ry}$ as a "commitment" to the randomness; and $C = g^{r(x+Mxy)}$ as the signature on the message M.

An aggregate in our case consists of four group elements A, B, C, D where the first three elements A, B, C play the same role as in the standard signature scheme. The element D serves as a programmable "memory". We illustrate the necessity for this element on the following example, where a second user wishes to aggregate his signature on a message M_2 onto the aggregate $\sigma_1 = (A_1, B_1, C_1)$. Let $(sk_i, pk_i) = ((x_i, y_i), (g^{x_i}, g^{y_i}))$ denote the key-pair of the i-th user (here for $i = 1, 2$) and let M_2 be the message to be signed. This second user first views $A_1 = g^r$ as the randomness to be used for his signature. It then computes $B \leftarrow B_1 \cdot A_1^{y_2} = g^{r(y_1+y_2)}$ and $C \leftarrow C_1 \cdot A_1^{x_2} A_1^{M_2 x_2 y_2} = g^{r(x_1+M_1 x_1 y_1+x_2+M_2 x_2 y_2)}$. If we now wish to verify that both signatures are valid, we must to check that:

$$\prod_i \left(\mathbf{e}(X_i, A) \cdot \mathbf{e}(X_i, B)^{M_i}\right) = \mathbf{e}(g, A)^{x_1+x_2} \cdot \mathbf{e}(g, B)^{M_1 x_1 + M_2 x_2} = \mathbf{e}(g, C)$$

holds. Unfortunately, this is not the case, because an "error" occurs during the evaluation of the second bilinear map:

$$\begin{aligned}
\mathbf{e}(g, B)^{M_1 x_1 + M_2 x_2} &= \mathbf{e}(g, B)^{M_1 x_1} \cdot \mathbf{e}(g, B)^{M_2 x_2} \\
&= \mathbf{e}(g, A^{y_1+y_2})^{M_1 x_1} \cdot \mathbf{e}(g, A^{y_1+y_2})^{M_2 x_2} \\
&= \mathbf{e}(g, A)^{M_1 y_1 x_1 + M_1 y_2 x_1} \cdot \mathbf{e}(g, A)^{M_2 y_1 x_2 + M_2 y_2 x_2}\,.
\end{aligned}$$

In fact, the evaluation results in two "noisy" elements: $M_1 y_2 x_1$ and $M_2 y_1 x_2$. We now program the element D such that it cancels all these elements out.

3.4 The Construction

An aggregate $\sigma := (A, B, C, D)$ on messages $\mathbf{M} := (M_1, \ldots, M_\ell)$ under public keys $\mathbf{pk} := (pk_1, \ldots, pk_\ell)$ has the form:

$$A = g^r \quad ; \quad B = \prod_i g^{r\, y_i} \quad ; \quad C = \prod_i g^{r(x_i + M_i x_i y_i)} \quad ; \quad D = \prod_{i \neq j} g^{M_i x_i y_j} \quad ;$$

where A may be seen as some "shared" randomness. The scheme $\mathsf{SAS} = (\mathsf{SeqPKg}, \mathsf{SeqKg}, \mathsf{SeqSign}, \mathsf{SeqVf})$ is defined as follows:

Parameter Generation. $\mathsf{PKg}(1^\lambda)$ chooses two suitable groups $\mathbb{G}, \mathbb{G}_T$, a generator $g \in \mathbb{G}$ and returns $I = (\mathbb{G}, \mathbb{G}_T, g, \mathbf{e})$.

Key Generation. For a particular signer, algorithm $\mathsf{SeqKg}(I)$ picks $x \leftarrow \mathbb{Z}_p$ and $y \leftarrow \mathbb{Z}_p$ and sets $X \leftarrow g^x$ as well as $Y \leftarrow g^y$. It returns sk as (I, x, y) and pk as $pk = (I, X, Y)$.

Sequential Signing. Algorithm SeqAgg takes as input a secret signing key $sk = (I, x, y)$, a message $M \in \mathbb{Z}_p$, an aggregate-so-far σ', a sequence of messages $\mathbf{M} = (M_1, \ldots . M_i)$, and a sequence of public keys $\mathbf{pk} = (pk_1, \ldots, pk_i)$. The algorithm first checks that $|\mathbf{M}| = |\mathbf{pk}|$ and that $\mathsf{SeqVf}(\sigma', \mathbf{M}, \mathbf{pk}) = 1$. If so, it parses σ' as (A', B', C', D'), and it sets

$$w_1 \leftarrow A' \quad ; \quad w_2 \leftarrow B' \cdot (A')^y \quad ; \quad w_3 \leftarrow C' \cdot (A')^{x + Mxy} \quad ;$$

$$\text{and} \quad w_4 \leftarrow D' \cdot \left(\prod_{j=1}^{i} X_j^{y M_j} \cdot Y_j^{xM} \right) .$$

Note that the first signer uses $(1, 1, 1, 1)$ as the aggregate-so-far. The tuple (w_1, w_2, w_3, w_4) forms a valid aggregate signature (for our algorithm) on messages $\mathbf{M}||M$ under public keys $\mathbf{pk}||pk$. Now, we have to re-randomize the aggregate, or an attacker could forge [2]:

$$\tilde{r} \leftarrow \mathbb{Z}_p^* \quad ; \quad A \leftarrow w_1^{\tilde{r}} \quad ; \quad B \leftarrow w_2^{\tilde{r}} \quad ; \quad C \leftarrow w_3^{\tilde{r}} \quad ; \quad D \leftarrow w_4 .$$

It follows easily that $\sigma = (A, B, C, D)$ is also a valid aggregate on messages $\mathbf{M}||M$ under public keys $\mathbf{pk}||pk$ with randomness $g^{r\,\tilde{r}}$.

[2] In particular, consider an adversary that chooses some random keys (x, y), a message M, and a randomness r. It computes the signature as an honest signer and sends it as an aggregate-so-far to the oracle (having the secret keys (x_c, y_c)) together with a challenge message M_c. If the oracle does not re-randomize the aggregate, then the adversary receives an element $C = g^{r(x + Mxy + x_c + M_c x_c y_c)}$. As the adversary knows the randomness r, it can easily obtain the value $XY_c = g^{x_c y_c}$ and can forge a signature on an arbitrary message.

Aggregate Verification. The input of algorithm SeqVf consists of a sequence of public keys $\mathbf{pk} = (pk_1, \ldots, pk_\ell)$, a sequence of message $\mathbf{M} = (M_1, \ldots, M_\ell)$ and an aggregate σ. It parses the aggregate σ as (A, B, C, D). It first checks that $|\mathbf{M}| = |\mathbf{pk}|$ and that no public key appears twice in $\mathbf{pk}$. Afterwards, it validates the structure of the elements A, B, and D:

$$\mathbf{e}(A, \prod_i Y_i) = \mathbf{e}(g, B) \quad \text{and} \quad \prod_{i \neq j} \mathbf{e}(X_i, Y_j)^{M_i} = \mathbf{e}(g, D),$$

and proves that C is formed correctly:

$$\prod_i \left(\mathbf{e}(X_i, A) \cdot \mathbf{e}(X_i, B)^{M_i}\right) \cdot \mathbf{e}(A, D)^{-1} = \mathbf{e}(g, C)\,.$$

If all equations are valid, SeqVf outputs 1; otherwise it returns 0.

Completeness follows inductively.

Performance and Symmetric Verification. The verification equation of element C suggests that, for N signers, a total number of $3\,N$ pairings have to be computed. However, this is not the case. We chose to present the computation in this form for the sake of esthetics. A more computationally efficient version of the verification equation is the following:

$$\mathbf{e}(\prod_i X_i, A) \cdot \mathbf{e}(\prod_i X_i^{M_i}, B) \cdot \mathbf{e}(A, D)^{-1} = \mathbf{e}(g, C)$$

which is identical to the one above, but which requires the computation of only three pairings irrespective of the number of signers. A more efficient variant for the verification of the element D is the following

$$\prod_i \mathbf{e}(X_i^{M_i}, \prod_j Y_i) = \mathbf{e}(g, D).$$

This equation involves only n pairing computations (instead of $n^2/2$).

Our construction has the additional feature, similar to [10,27], that the verifier does not need to know the order in which the aggregate was created. This property, sometimes called symmetric verification, is useful for several applications such as, e.g., building an optimistic fair exchange out of any sequential two-party multisignature as shown by Dodis et al. [17].

Applications to Ordered Multisignature. In a multisignature scheme a group of signers sign the *same* message such that the signature has roughly the same size as an ordinary signature. Recently, Boldyreva, Gentry, O'Neill, and Yum generalized multisignatures to *ordered multisignatures* (OMS). This primitives has the additional property that the adversary cannot re-order the position of honest signers [7]. The authors also suggest a generic transformation that turns every aggregate signature scheme into an orderd multisignature scheme. The idea is to encode the position of each signer into the message, i.e.,

suppose the the signer is at position i and wishes to sign the common messsage M, then it runs the aggregate signing algorithm on the message $M\|i$. Applying this transformation to our scheme yields also the first OMS with short keys in the standard model.

3.5 Proof of Security

In this section we prove the security of our scheme.

Theorem 1. *The aggregate signature scheme is $(t, q_C, q_S, n, \epsilon)$-secure with respect to Definition 4, if the CL-signature scheme (t', q', ϵ') unforgeable on $\mathbb{G}$, where*

$$t' = t + O(q_C + nq_S + n) \quad \text{and} \quad q' = q_S \quad \text{and} \quad \epsilon' = \epsilon\,.$$

Proof. We use a proof by contradiction. Suppose that $\mathcal{A}$ is an adversary which forges the sequential aggregate signature scheme. We then show how to construct an algorithm $\mathcal{B}$ that breaks the unforgeability of the underlying CL-signature scheme.

Setup. The algorithm $\mathcal{B}$ gets as input a public-key $pk_c = (I, X, Y)$. It initializes the query list $C \leftarrow \emptyset$ and runs a black-box simulation of the attacker $\mathcal{A}$ on input pk_c.

Certification Queries. If algorithm $\mathcal{A}$ wishes to certify a public key $pk = (I, X, Y)$, it hands over the corresponding secret key $sk = (I, x, y)$. Algorithm $\mathcal{B}$ verifies that (x, y) is the private key corresponding to pk. If so, it adds (sk, pk) to the list of certified keys, i.e., $C \leftarrow C \cup (sk, pk)$. Otherwise, it rejects the query.

Signature Queries. Whenever the algorithm $\mathcal{A}$ invokes its aggregate signing oracle $\mathsf{SeqAgg}(sk, \cdot)$ on: a message M, an aggregate-so-far σ', a sequence of messages $\mathbf{M}'$, and a sequence of public keys $\mathbf{pk}'$, then algorithm $\mathcal{B}$ behaves as follows: it first checks that σ' is a valid aggregate on messages $\mathbf{M}'$ under public keys $\mathbf{pk}'$; afterwards, it checks that all public keys $pk \in \mathbf{pk}$ are certified, that each key appears only once, and that $|\mathbf{pk}'| = |\mathbf{M}'|$. If any of these conditions is violated, then $\mathcal{B}$ returns `invalid`. In the following let $|\mathbf{M}'| = |\mathbf{pk}'| =: q$.

Otherwise, algorithm $\mathcal{B}$ invokes its own signing oracle $\mathsf{Sig}(sk_c, \cdot)$ on M and receives the output signature σ. Note that σ is also an aggregate on message M under the public key pk_c. Next, algorithm $\mathcal{B}$ "adds" the missing q signatures onto the aggregate. More precisely, it sets: $\sigma_0 \leftarrow \sigma$, $M_0 \leftarrow M$, and $pk_0 \leftarrow pk_c$, and it executes $\sigma_i \leftarrow \mathsf{SeqAgg}(sk_i, M_i, \sigma_{i-1}, \mathbf{M}_{i-1}, \mathbf{pk}_{i-1})$, where $\mathbf{M}_{i-1} = (M_1, \ldots, M_{i-1})$ and $\mathbf{pk}_{i-1} = (pk_1, \ldots, pk_{i-1})$ for $i = 1, \ldots, q$. Observe that this is possible since all public keys in $\mathbf{pk}$ are registered, and that each private key is stored in C. Afterwards, $\mathcal{B}$ returns the aggregate $\sigma \leftarrow \sigma_i$ on messages $\mathbf{M} \leftarrow \mathbf{M}'||M$ under public keys $\mathbf{pk} \leftarrow \mathbf{pk}'||pk$.

Output. Finally $\mathcal{A}$ stops, eventually outputting a valid forgery σ' on messages $\mathbf{M}^*$ under public keys $\mathbf{pk}^*$. This forgery is valid if the following relations hold:

– $|\mathbf{pk}^*| = |\mathbf{M}^*|$;
– $\mathsf{SeqVf}(\sigma^*, \mathbf{M}^*, \mathbf{pk}^*) = 1$,
– $pk_c \in \mathbf{pk}^*$ say at index i_c,
– all keys in $\mathbf{pk}^*$ except for the challenge key pk_c are registered,
– $\mathcal{A}$ never queried its sequential signing oracle about the message M_{i_c}.

W.l.o.g. we assume that the challenge key pk_c is stored at the first position in $\mathbf{pk}^*$, i.e., $i_c = 1$ and let $|\mathbf{pk}^*| =: q$. Just as in the case of [27], this is not a restriction, because the verification equation does not take into account the order of the participants.

Next, for each $2 \leq i \leq q$ let (I, x_i, y_i) be the secret key corresponding to $pk_i = (I, X_i, Y_i)$. Algorithm $\mathcal{B}$ computes:

$$a^* \leftarrow A^* \; ; \; b^* \leftarrow B^* \cdot \left((A^*)^{\sum_{i=2}^{q} y_i}\right)^{-1} \; ; \quad c^* \leftarrow C^* \cdot \left((A^*)^{\sum_{i=2}^{q} x_i + M_i x_i y_i}\right)^{-1},$$

and outputs $(M^*, \sigma^*) \leftarrow (M_1, (a^*, b^*, c^*))$.

For the analysis, first note that $\mathcal{B}$ is efficient since $\mathcal{A}$ runs in polynomial-time and since the attacker $\mathcal{B}$ performs a perfect simulation from $\mathcal{A}$'s point of view. In the following, we show that $\mathcal{B}$ succeeds whenever $\mathcal{A}$ does. Thus, we have to show firstly that algorithm $\mathcal{B}$ outputs a valid CL-signature, and secondly, that $\mathcal{B}$ has never queried the message to its signing oracle. The second condition follows easily from the assumption that $\mathcal{A}$ outputs a valid forgery. For the first property we show that we can divide all other signatures out such that only the signature for the challenge key remains. Consider the first verification equation of the aggregate signature scheme. We know that:

$$\mathbf{e}(A^*, \prod_i Y_i) = \mathbf{e}(g^r, \prod_i g^{y_i}) = \mathbf{e}(g, \prod_i g^{r y_i}) = \mathbf{e}(g, \prod_i (A^*)^{y_i}) = \mathbf{e}(g, B^*) \quad (1)$$

for some r; this implies that:

$$\begin{aligned}
\mathbf{e}(g, b^*) &= \mathbf{e}\left(g, B^* \cdot \left((A^*)^{\sum_{i=2}^{q} y_i}\right)^{-1}\right) = \mathbf{e}\left(g, B^*\right) \cdot \mathbf{e}\left(g, \left((A^*)^{\sum_{i=2}^{q} y_i}\right)^{-1}\right) \\
&\overset{(1)}{=} \mathbf{e}(A^*, \prod_{i=1}^{q} Y_i) \cdot \mathbf{e}\left(g, \left((A^*)^{\sum_{i=2}^{q} y_i}\right)^{-1}\right) \\
&= \mathbf{e}(A^*, g^{\sum_{i=1}^{q} y_i})) \cdot \mathbf{e}\left(A^*, \left(g^{\sum_{i=2}^{q} y_i}\right)^{-1}\right) \\
&= \mathbf{e}(A^*, g^{\sum_{i=1}^{q} y_i})) \cdot \mathbf{e}(A^*, (g^{\sum_{i=2}^{q} y_i})^{-1}) = \mathbf{e}(A^*, g^{\sum_{i=1}^{q} y_i} g^{\sum_{i=2}^{q} -y_i}) \\
&= \mathbf{e}(a^*, Y_1)\,.
\end{aligned}$$

Now, the second and third verification equations of the aggregate signature scheme prove that:

$$\begin{aligned}
\prod_{i \neq j} \mathbf{e}(X_i, Y_j)^{M_i^*} &= \mathbf{e}(g, g)^{\sum_{i \neq j} M_i^* x_i y_j} = \mathbf{e}(g, g^{\sum_{i \neq j} M_i^* x_i y_j}) \\
&= \mathbf{e}(g, \prod_{i \neq j} g^{M_i^* x_i y_j}) = \mathbf{e}(g, D^*)
\end{aligned}$$

and that:

$$\prod_i \left[\mathbf{e}(X_i, A^*) \cdot \mathbf{e}(X_i, B^*)^{M_i^*}\right] \cdot \mathbf{e}(A^*, D^*)^{-1} = \mathbf{e}(g, C^*)\,. \tag{2}$$

This implies that:

$$\begin{aligned}\mathbf{e}(g, c^*) &= \mathbf{e}(g, C^*) \cdot \mathbf{e}\left(g, \left((A^*)^{\sum_{i=2}^{q} x_i + M_i^* x_i y_i}\right)^{-1}\right) \\ &\overset{(2)}{=} \prod_i \left[\mathbf{e}(X_i, A^*) \cdot \mathbf{e}(X_i, B^*)^{M_i^*}\right] \cdot \\ &\qquad \cdot \mathbf{e}(A^*, D^*)^{-1} \cdot \mathbf{e}\left(g, \left((A^*)^{\sum_{i=2}^{q} x_i + M_i^* x_i y_i}\right)^{-1}\right)\,.\end{aligned}$$

Observe that $\mathbf{e}(g, B^*) \overset{(1)}{=} \mathbf{e}(g, \prod_i (A^*)^{y_i}) = \mathbf{e}(g, \prod_i g^{r y_i})$ and thus,

$$\prod_i \mathbf{e}(X_i, B^*)^{M_i^*} = \prod_i \mathbf{e}(g, B^*)^{M_i^* x_i} = \prod_i \mathbf{e}(g, \prod_i g^{r y_i})^{M_i^* x_i}\,. \tag{3}$$

Next, we replace the index i of the product $\prod_i g^{r y_i}$ with j. Using the results of these equations, we have the following:

$$\begin{aligned}\mathbf{e}(g, c^*) &= \prod_i \left[\mathbf{e}(X_i, A^*) \cdot \mathbf{e}(X_i, B^*)^{M_i^*}\right] \cdot \\ &\qquad \cdot \mathbf{e}(A^*, D^*)^{-1} \cdot \mathbf{e}\left(g, \left((A^*)^{\sum_{i=2}^{q} x_i + M_i^* x_i y_i}\right)^{-1}\right) \\ &\overset{(3)}{=} \prod_i [\mathbf{e}(A^*, X_i)] \cdot \prod_i \left[\mathbf{e}(g, \prod_j g^{r y_j})^{M_i^* x_i}\right] \cdot \mathbf{e}(A^*, \prod_{i \neq j} g^{M_i^* x_i y_j})^{-1} \cdot \\ &\qquad \cdot \mathbf{e}\left(A^*, \left(g^{\sum_{i=2}^{q} x_i + M_i^* x_i y_i}\right)^{-1}\right) \\ &= \prod_i [\mathbf{e}(A^*, X_i)] \cdot \mathbf{e}(A^*, \prod_i \prod_j g^{M_i^* x_i y_j}) \cdot \mathbf{e}(A^*, \prod_{i \neq j} g^{M_i^* x_i y_j})^{-1} \cdot \\ &\qquad \cdot \mathbf{e}\left(A^*, \left(g^{\sum_{i=2}^{q} x_i + M_i^* x_i y_i}\right)^{-1}\right)\,.\end{aligned}$$

In the following, we divide the products $\mathbf{e}(A^*, \prod_i \prod_j g^{M_i^* x_i y_j})$ into two parts; the first part consists of all factors for which $i = j$, while the second part contains the remaining factors:

$$
\begin{aligned}
\mathbf{e}(g, c^*) &= \prod_i [\mathbf{e}(A^*, X_i)] \cdot \mathbf{e}(A^*, \prod_{i=j} g^{M_i^* x_i y_j} \prod_{i \neq j} g^{M_i^* x_i y_j}) \cdot \\
&\qquad \mathbf{e}(A^*, \prod_{i \neq j} g^{M_i^* x_i y_j})^{-1} \cdot \mathbf{e}\left(A^*, \left(g^{\sum_{i=2}^{q} x_i + M_i^* x_i y_i}\right)^{-1}\right) \\
&= \prod_i [\mathbf{e}(A^*, X_i)] \cdot \mathbf{e}(A^*, \prod_{i=j} g^{M_i^* x_i y_j}) \cdot \\
&\qquad \cdot \mathbf{e}(A^*, \prod_{i \neq j} g^{M_i^* x_i y_j}) \cdot \mathbf{e}(A^*, \prod_{i \neq j} g^{M_i^* x_i y_j})^{-1} \cdot \mathbf{e}\left(A^*, \left(g^{\sum_{i=2}^{q} x_i + M_i^* x_i y_i}\right)^{-1}\right) \\
&= \prod_{i=1}^{q} \left[\mathbf{e}(A^*, X_i) \cdot \mathbf{e}(A^*, g^{M_i^* x_i y_i})\right] \cdot \prod_{i=2}^{q} \mathbf{e}\left(A^*, \left(X_i \cdot g^{M_i^* x_i y_i}\right)^{-1}\right) \\
&= \prod_{i=1}^{q} \left[\mathbf{e}(A^*, X_i) \cdot \mathbf{e}(A^*, g^{M_i^* x_i y_i})\right] \cdot \prod_{i=2}^{q} \left[\mathbf{e}\left(A^*, (X_i)^{-1}\right) \cdot \mathbf{e}\left(A^*, \left(g^{M_i^* x_i y_i}\right)^{-1}\right)\right] \\
&= \mathbf{e}(A^*, X_1) \cdot \mathbf{e}(A^*, g^{M_1^* x_1 y_1}) \cdot \prod_{i=2}^{q} \left[\mathbf{e}(A^*, X_i) \cdot \mathbf{e}(A^*, g^{M_i^* x_i y_i})\right] \cdot \\
&\qquad \cdot \prod_{i=2}^{q} \left[\mathbf{e}\left(A^*, (X_i)^{-1}\right) \cdot \mathbf{e}\left(A^*, \left(g^{M_i^* x_i y_i}\right)^{-1}\right)\right] \\
&= \mathbf{e}(A^*, X_1) \cdot \mathbf{e}(A^*, g^{M_1^* x_1 y_1}) \cdot \prod_{i=2}^{q} \left[\mathbf{e}(A^*, X_i) \cdot \mathbf{e}(A^*, g^{M_i^* x_i y_i})\right] \cdot \\
&\qquad \cdot \prod_{i=2}^{q} \left[\mathbf{e}\left(A^*, (X_i)\right)^{-1} \cdot \mathbf{e}\left(A^*, \left(g^{M_i^* x_i y_i}\right)\right)^{-1}\right] \\
&= \mathbf{e}(A^*, X_1) \cdot \mathbf{e}(A^*, g^{M_1^* x_1 y_1}) \cdot \prod_{i=2}^{q} \left[\mathbf{e}(A^*, X_i) \cdot \mathbf{e}(A^*, g^{M_i^* x_i y_i})\right] \cdot \\
&\qquad \cdot \prod_{i=2}^{q} \left[\mathbf{e}\left(A^*, (X_i)\right) \cdot \mathbf{e}\left(A^*, \left(g^{M_i^* x_i y_i}\right)\right)\right]^{-1} \\
&= \mathbf{e}(A^*, X_1) \cdot \mathbf{e}(A^*, g^{M_1^* x_1 y_1}) \\
&\qquad = \mathbf{e}(A^*, X_1) \cdot \mathbf{e}(g^r, g^{M_1^* x_1 y_1}) \cdot \mathbf{e}(A^*, X_1) \cdot \mathbf{e}(g^{x_1}, g^{r y_1})^{M_1^*} \\
&= \mathbf{e}(a^*, X_1) \cdot \mathbf{e}(X_1, b^*)^{M_1^*}
\end{aligned}
$$

as desired. Thus, $\mathcal{B}$ succeeds whenever $\mathcal{A}$ does.

Acknowledgments. The author thanks Heike Schröder, Cristina Onete, Markus Rückert, and the anonymous reviewers for valuable comments. This work was partially supported by the US Army Research Laboratory and the UK Ministry of Defence under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the US Government, the UK Ministry of

References

1. Ateniese, G., Camenisch, J., de Medeiros, B.: Untraceable rfid tags via insubvertible encryption. In: Proceedings of the Annual Conference on Computer and Communications Security (CCS), pp. 92–101. ACM, New York (2005)
2. 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)
3. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: ACM Conference on Computer and Communications Security 2006, pp. 390–399. ACM Press, New York (2006)
4. Bender, A., Katz, J., Morselli, R.: Ring signatures: Stronger definitions, and constructions without random oracles. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 60–79. Springer, Heidelberg (2006)
5. Boldyreva, A.: Efficient 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)
6. Boldyreva, A., Gentry, C., O'Neill, A., Yum, D.H.: Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In: Proceedings of the Annual Conference on Computer and Communications Security (CCS 2007), pp. 276–285. ACM Press, New York (2007)
7. Boldyreva, A., Gentry, C., O'Neill, A., Yum, D.H.: New multiparty signature schemes for network routing applications. ACM Transactions on Information and System Security (TISSEC) 12(1) (2008)
8. Boneh, D., Boyen, X.: Short signatures without random oracles and the sdh assumption in bilinear groups. Journal of Cryptology 21(2), 149–177 (2008)
9. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004)
10. 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)
11. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001)
12. Camenisch, J., Hohenberger, S., Kohlweiss, M., Lysyanskaya, A., Meyerovich, M.: How to win the clonewars: Efficient periodic n-times anonymous authentication. In: Proceedings of the Annual Conference on Computer and Communications Security (CCS), pp. 201–210. ACM Press, New York (2006)
13. Camenisch, J., Hohenberger, S., Pedersen, M.Ø.: Batch verification of short signatures. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 246–263. Springer, Heidelberg (2007)

14. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Heidelberg (2004)
15. Chatterjee, S., Hankerson, D., Knapp, E., Menezes, A.: Comparing two pairing-based aggregate signature schemes. Cryptology ePrint Archive, Report 2009/060 (2009), http://eprint.iacr.org/
16. Damgård, I., Dupont, K., Pedersen, M.Ø.: Unclonable group identification. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 555–572. Springer, Heidelberg (2006)
17. Dodis, Y., Lee, P.J., Yum, D.H.: Optimistic fair exchange in a multi-user setting. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 118–133. Springer, Heidelberg (2007)
18. Eikemeier, O., Fischlin, M., Götzmann, J.F., Lehmann, A., Schröder, D., Schröder, P., Wagner, D.: History-free aggregate message authentication codes. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 309–328. Springer, Heidelberg (2010)
19. Fischlin, M., Lehmann, A., Schröder, D.: History-free sequential aggregate signatures. Cryptology ePrint Archive, Report 2011/231 (2011), http://eprint.iacr.org/
20. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. 17(2), 281–308 (1988)
21. Harn, L.: Group-oriented (t, n) threshold digital signature scheme and digital multisignature. IEE Proceedings of Computers and Digital Techniques 141(5), 307–313 (1994)
22. Hofheinz, D., Kiltz, E.: Programmable hash functions and their applications. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 21–38. Springer, Heidelberg (2008)
23. IOC: Ioc tsunami website (2009), http://ioc3.unesco.org/, http://ioc3.unesco.org/indotsunami/
24. Itakura, K., Nakamura, K.: A public key cryptosystem suitable for digital multisignatures. NEC Research & Development 71, 1–8 (1983)
25. Katz, J., Lindell, A.Y.: Aggregate message authentication codes. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 155–169. Springer, Heidelberg (2008)
26. Kiayias, A., Zhou, H.-S.: Concurrent blind signatures without random oracles. In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 49–62. Springer, Heidelberg (2006)
27. 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)
28. 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)
29. Lysyanskaya, A., Rivest, R.L., Sahai, A., Wolf, S.: Pseudonym systems. In: Walker, M. (ed.) Cryptography and Coding 1999. LNCS, vol. 1746, pp. 184–199. Springer, Heidelberg (1999)
30. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures: extended abstract. In: ACM Conference on Computer and Communications Security 2001, pp. 245–254. ACM Press, New York (2001)

31. Neven, G.: Efficient sequential aggregate signed data. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 52–69. Springer, Heidelberg (2008)
32. Ohta, K., Okamoto, T.: A digital multisignature scheme based on the fiat-shamir scheme. In: Matsumoto, T., Imai, H., Rivest, R.L. (eds.) ASIACRYPT 1991. LNCS, vol. 739, pp. 139–148. Springer, Heidelberg (1993)
33. Okamoto, T.: A digital multisignature schema using bijective public-key cryptosystems. ACM Trans. Comput. Syst. 6(4), 432–441 (1988)
34. Ristenpart, T., Yilek, S.: The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 228–245. Springer, Heidelberg (2007)
35. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997)