# History-Free Sequential Aggregate Signatures

Marc Fischlin[1], Anja Lehmann[2], and Dominique Schröder[3]

[1] Darmstadt University of Technology, Germany
[2] IBM Research Zurich, Switzerland
[3] University of Maryland, USA & Saarland University, Germany

**Abstract.** Aggregation schemes allow to combine several cryptographic values like message authentication codes or signatures into a shorter value such that, despite compression, some notion of unforgeability is preserved. Recently, Eikemeier et al. (SCN 2010) considered the notion of *history-free* sequential aggregation for message authentication codes, where the sequentially-executed aggregation algorithm does not need to receive the previous messages in the sequence as input. Here we discuss the idea for signatures where the new aggregate does not rely on the previous messages and public keys either, thus inhibiting the costly verifications in each aggregation step as in previous schemes by Lysyanskaya et al. (Eurocrypt 2004) and Neven (Eurocrypt 2008). Analogously to MACs we argue about new security definitions for such schemes and compare them to previous notions for history-dependent schemes. We finally give a construction based on the BLS signature scheme which satisfies our notion.

## 1 Introduction

Aggregate signature schemes [6] allow to combine multiple signatures from different senders for possibly different messages, such that the aggregate has roughly the same size as a single signature. This helps to reduce the communication overhead in settings where authenticated information is forwarded from one party to another, such as the S-BGP routing protocol or certificate chains [6,13,3,5]. As in the case of regular signature schemes, the validity of aggregates can be publicly verified given all messages and public keys.

The original proposal of Boneh at al. [6] supports aggregation of the data independently of the order of the parties and, furthermore, the aggregating algorithm only relies on the aggregates and public data. In contrast, most other solutions today like [14,13,5,3,16,17] are *sequential* aggregate schemes where each party derives the next aggregate by taking the private key, the previous aggregate, and all the previous messages together with the corresponding keys in the sequence into account. For instance, in all[1] known sequential signature schemes the aggregation algorithm first checks with the public keys that the current aggregate

[1] With the exception of the recent work by Brogle et al. [8], discussed at the end of the introduction.

is a valid signature for the preceding message sequence. Often, they also incorporate these messages in the computation of the new aggregate. Thus, so far, the aggregation in sequential signature schemes seems to be much more expensive than in the non-sequential setting, which might render sequential schemes impractical for resource-constraint devices. Another issue, pointed out in [8], is that the verification requires also obtaining and checking the public keys of the users in the sequence.

## 1.1 History-Free Sequential Aggregation

Recently, Eikemeier et al. [10] introduced the notion of *history-freeness* in the context of aggregate MACs, which aims to preserve the "lightweight" aggregation approach from general aggregate schemes also in the sequential setting. More precisely, in a history-free MAC a new aggregate is derived only from the aggregate-so-far and the local message, but does not rely on (explicit) access to the previous messages. Note that, strictly speaking, the aggregate-so-far certainly contains some information about the previous messages; this information, however, is limited due to the size restriction for aggregates.

In this work we adopt the notion of history-freeness to the case of sequential aggregate *signatures*, only allowing the aggregate-so-far, the local message, and signing key to enter the computation, but not the previous messages and public keys in the sequence. For signatures this property is especially worthwhile, because it means that the costly signature verifications for each aggregation step are suppressed. In fact, since the security of previous schemes strongly relies on such checks, omitting them indicates the hardness of finding history-free schemes. Eikemeier et al. [10] achieve this, to some extent, for the case of MACs by using an underlying pseudorandom permutation to encrypt parts of the data. This is usually not an admissible strategy for the case of signatures.

At first, history-free sequential aggregation might seem to be the second best solution compared to non-ordered aggregation (with history-free aggregation quasi built in). However, sequential aggregation is required for many applications such as for authenticating routing information or for certificate chains, and in these applications the verifiability of the order of signing steps is usually important, whereas general aggregate schemes do not allow this. Following the terminology for multi-signatures [5] we call such schemes ordered sequential-aggregate schemes. We also remark that all known sequential aggregate schemes are ordered, except for the one by Lu et al. [13], and that we usually consider history-freeness only in connection with such ordered schemes.

## 1.2 New Security Models

Introducing the idea of history-freeness affects known security definitions for sequential signature schemes. Since the history of previously signed messages is not available to the aggregation algorithm, an adversary can now initiate aggregation chains "from the middle", without specifying how the initial message sequence looks like. The starting aggregate for such a truncated iteration does not even need to be valid, as checking the validity of the aggregate with respect to the preceding message sequence is impossible for the aggregation algorithm.

Our security notions for history-free schemes, adopted from the work by Eikemeier et al. [10], follow the well-known approach for (regular and aggregate) signatures that an adversary can request data via oracles and is supposed to eventually output a valid but non-trivial forgery. In the original LMRS security model for sequential aggregation with full information about preceding messages [14], the adversary is considered to win if it produces a valid aggregate for a non-trivial sequence, where trivial sequences are previously queried sequences and, since appending some iterations for controlled parties is easy for the adversary, such extended sequences thereof.

Specifying the trivial combinations in our history-free model is more delicate because the adversary now gets to query partial chains and can potentially glue several of these data together. We resolve this by following the approach of Eikemeier et al., that is, by defining a transitive closure of trivial sequences, consisting of matching combinations of (possibly many) previously seen aggregates and contributions by corrupt parties. We define two versions of this closure, depending on whether intermediate values of partial chains are available to the adversary or not, yielding two security notions (one being stronger and implying the other). Intuitively, due to the additional adversarial power, one would expect our new security models to be weaker than the original ones for sequential aggregation. Interestingly, though, both our security notions for history-freeness are strictly stronger than the security model for sequential aggregation due to Lu et al. [13], but incomparable to the one of Lysyanskaya et al. [14], as we show in Section 3.3. Even more remarkably, by slightly relaxing the requirement for history-freeness, we can easily achieve the [14] security property (on top of our aggregation-unforgeability notion) if we simply prepend the hash value of all previous public keys and messages in the sequence to the message to be signed next. By this we get a strongly secure sequential aggregate signature which does not need verification of all preceding signatures!

We also briefly revisit the case of non-ordered aggregates. Here, adapting the idea of the closure yields strictly stronger security guarantees than in previous definitions for non-sequential schemes. Our models, both for sequential and for non-ordered schemes, reflect the resistance of aggregate schemes against "mix-and-match" attacks, where an attacker is already considered successful if it can recombine learned aggregates into a "fresh" aggregate that it has not seen before, or is able to remove parts of the aggregates. This is opposed to the common approach of reducing the unforgeability of aggregation schemes to the unforgeability of individual messages, where combining aggregates or removing a party's contribution are not deemed to be successful attacks (because they do not forge an individual signature). This is discussed for the symmetric setting in more detail in [10]. Yet, we are not aware if that high security standard can be achieved for aggregate signatures. Nonetheless, as a side effect of our approach, we point out that the scheme by Boneh et al. [6] allows attacks which are not covered by their security models. The discussion appears in Section 5.

### 1.3 Building History-Free Schemes

We finally provide a solution meeting our requirements in Section 4. We give a construction based on the signature scheme of Boneh et al. [7], which has already been successfully transformed into the BGLS scheme for non-sequential aggregation [6]. By this we derive a scheme for history-free sequential aggregation. Observe again that the resulting scheme also comes with the verifiability of the aggregation order.

Our construction chains the aggregates with the help of a collision-resistant hash function, i.e., instead of signing only the local message, we first compute the hash value of this message together with the previous aggregate.[2] Hence, instead of verifying a chain of signatures our aggregation algorithm only needs to compute bilinear mappings. The aggregates of our scheme are slightly larger than the ones of the original BGLS scheme and the construction satisfies our weaker security notion.

### 1.4 Concurrent Work

Recently, Brogle et al. [8] proposed a notion of sequential aggregate signatures with so-called lazy verification, resembling the idea of history-freeness as defined in [10] and also used here closely. They designed and implemented a history-free scheme based on trapdoor permutations, with a special focus on the BGPsec protocol [12]. Their security model, albeit appropriate for the BGPsec case, is a relaxation of the LMRS model which is implied by (even the weaker version of) our security notion. The reason is roughly that this relaxation merely demands that the message in the forgery has not been signed by the honest user before, implying that it cannot be in the closure and therefore also constitutes a breach of security in our model. We note that the relaxed LMRS notion does not cover the class of mix-and-match attacks discussed in [10] and here. The construction in [8] produces signatures proportional to the number of signers and explicitly relies on the random oracle model. In contrast, our scheme generates signatures of size independent of the number of signers, only implicitly relies on the random oracle model through the currently best proof for the underlying BLS signature scheme in the random oracle model. Our solution comes with stronger unforgeability guarantees (under reasonable cryptographic assumptions).

## 2 Preliminaries

### 2.1 Sequential Aggregate Signature Schemes

An aggregate signature [6] is a single signature of different signers on different messages such that this aggregate has roughly the same size as an ordinary

[2] The tricky part here is that we do not use the aggregate as it is, but first apply the underlying bilinear mapping to it, before giving it to the hash function. This is necessary to allow verification of aggregates without seeing individual signatures and relies on specific properties of the BLS scheme.

signature. In the sequential case the aggregation algorithm gets as input a sequence of public keys $\mathbf{pk} = (pk_1, \ldots, pk_i)$ and messages $\mathbf{M} = (M_1, \ldots, M_i)$, an aggregate $\sigma'$ for this sequence, a message $M$ and the secret signing key $sk$ (with corresponding public key $pk$). It returns the new aggregate $\sigma$ for the sequence $\mathbf{pk}||pk := (pk_1, \ldots, pk_i, pk)$ and $\mathbf{M}||M := (M_1, \ldots, M_i, M)$. More formally:

**Definition 1 (Sequential Aggregate Signature Scheme).** *A sequential aggregate signature scheme is a tuple of efficient algorithms* $\mathsf{SAS} = (\mathsf{SeqKg}, \mathsf{SeqAgg}, \mathsf{SeqAggVf})$*, where*

**Key Generation.** $\mathsf{SeqKg}(1^n)$ *generates a key pair* $(sk, pk)$ *where pk is recoverable from sk.*

**Signature Aggregation.** *The aggregation algorithm* $\mathsf{SeqAgg}(sk, M, \sigma', \mathbf{M}, \mathbf{pk})$ *takes as input a secret key sk, a message* $M \in \{0,1\}^*$*, an aggregate* $\sigma'$ *and sequences* $\mathbf{M} = (M_1, \ldots, M_i)$ *of messages and* $\mathbf{pk} = (pk_1, \ldots, pk_i)$ *of public keys and computes the aggregate* $\sigma$ *for message sequence* $\mathbf{M}||M = (M_1, \ldots, M_i, M)$ *and key sequence* $\mathbf{pk}||pk = (pk_1, \ldots, pk_i, pk)$*. (We assume that there is a special "starting" symbol* $\sigma_0 = \emptyset$ *for the empty aggregate, different from all other possible aggregates.)*

**Aggregate Verification.** *The algorithm* $\mathsf{SeqAggVf}(\mathbf{pk}, \mathbf{M}, \sigma)$ *takes as input a sequence of public keys* $\mathbf{pk} = (pk_1, \ldots, pk_i)$*, a sequence of messages* $\mathbf{M} = (M_1, \ldots, M_i)$ *as well as an aggregate* $\sigma$*. It returns a bit.*

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

Note that we do not define "pure" signing and verification algorithms but only the aggregate counterparts. We can specify such algorithms in a straightforward way via the aggregation algorithm run on the starting aggregate $\sigma_0$. In fact, this is often how, vice versa, the aggregation algorithm works on this empty sequence. Second, we do not put any formal restriction on the size of aggregates, in the sense that aggregates must be smaller than individual signatures. Such restrictions can be always met by first "inflating" regular signatures artificially. We thus leave it to common sense to exclude such trivial examples. Finally, throughout the paper we assume that public keys of parties are unique, say, they include the identity and a sequence number as common in certificates.

## 2.2 LMRS Security of Sequential Aggregate Schemes

Lysyanskaya et al. [14] propose a security model for sequential aggregate signature schemes based on the chosen-key model of [4,6]. The adversary gets as input a challenge public key $pk_c$ and has access to a sequential aggregate signing oracle $\mathsf{SeqAgg}(sk_c, \cdots)$ which takes a message $M$, an aggregate $\sigma'$ and sequences $\mathbf{M}$ and $\mathbf{pk}$ as input and returns the new aggregate $\sigma$. The adversary wins if it manages to output a valid sequential aggregate signature for a sequence $\mathbf{M}^* = (M_1^*, \ldots, M_i^*)$

under public keys $\mathbf{pk}^* = (pk_1^*, \ldots, pk_i^*)$ and $\mathbf{pk}^*$ contains the challenge key $pk_c$ and the sequence $(M_1^*, \ldots, M_{i_c}^*)$ with $(pk_1^*, \ldots, pk_{i_c}^*)$ has never been queried to oracle $\mathsf{SeqAgg}$, where $i_c$ denotes the index of $pk_c$ in $\mathbf{pk}^*$.

For the sake of distinctiveness with the unforgeability notion for regular signature schemes we call schemes being immune against such adversaries *sequentially unforgeable*:

**Definition 2.** *A sequential aggregate signature scheme* $\mathsf{SAS} = (\mathsf{SeqKg}, \mathsf{SeqAgg}, \mathsf{SeqAggVf})$ *is* sequentially unforgeable *if for any efficient algorithm* $\mathcal{A}$ *the probability that the experiment* $\mathsf{SeqForge}_{\mathcal{A}}^{\mathsf{SAS}}$ *evaluates to* $1$ *is negligible (as a function of* $n$*), where*

```
Experiment $\mathsf{SeqForge}_{\mathcal{A}}^{\mathsf{SAS}}(n)$
    $(sk_c, pk_c), \leftarrow \mathsf{SeqKg}(1^n)$
    $(\mathbf{pk}^*, \mathbf{M}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{SeqAgg}(sk_c, \cdots)}(pk_c)$
    Let $i_c$ be the index of $pk_c$ in $\mathbf{pk}^* = (pk_1^*, \ldots, pk_\ell^*)$ and $\mathbf{M}^* = (M_1^*, \ldots, M_\ell^*)$.
    Return $1$ iff $\mathsf{SeqAggVf}(\mathbf{pk}^*, \mathbf{M}^*, \sigma^*) = 1$
        and $pk_c \in \mathbf{pk}^*$ and $pk_i \neq pk_j$ for $1 \leq i < j \leq \ell$ and
        $\mathcal{A}$ never queried $\mathsf{SeqAgg}(sk_c, \cdots)$ about $(M_1^*, \ldots, M_{i_c}^*)$, $(pk_1^*, \ldots, pk_{i_c}^*)$.
```

## 3 Security of History-Free Sequential Signatures

### 3.1 History-Freeness

So far, sequential aggregate schemes usually include the previous messages and public keys when deriving the new aggregate. This is a crucial disadvantage compared to the "lightweight" aggregation in non-sequential schemes, where the aggregation only depends on the previous signatures. To circumvent this issue we now apply the recently proposed notion of history-freeness [10] which restricts the input for the aggregation algorithm to the aggregate-so-far and the local message, i.e., the aggregation does not get access to the previous messages and keys. More formally:

**Definition 3 (History-Freeness).** *A sequential aggregate signature scheme* $\mathsf{SAS} = (\mathsf{SeqKg}, \mathsf{SeqAgg}, \mathsf{SeqAggVf})$ *is called* history-free *if there exists an efficient algorithm* $\mathsf{SeqAgg}_{hf}$ *such that* $\mathsf{SeqAgg}_{hf}(\cdot,\cdot,\cdot) = \mathsf{SeqAgg}(\cdot,\cdot,\cdot,\mathbf{M},\mathbf{pk})$ *for all* $\mathbf{M}, \mathbf{pk}$.

To save on notation we will often identify $\mathsf{SeqAgg}_{\mathsf{hf}}$ with $\mathsf{SeqAgg}$ and simply omit $\mathbf{M}, \mathbf{pk}$ from the input of $\mathsf{SeqAgg}$.

Note that history-free sequential signature schemes are not the same as non-sequential aggregate signatures as defined by Boneh et al. [6]. As mentioned in the introduction, the security requirement for (history-free) sequential schemes often allows to check the order of the signers, in contrast to non-sequential schemes.

## 3.2 Security Model

When considering history-free signature schemes the $\mathsf{LMRS}$ security model for sequential schemes [14] does not fully reflect the new conditions of the adversary and the desired security guarantees. This stems from the fact that in the history-free setting the previously signed messages are not available to the aggregation algorithm, which allows an adversary to trigger new aggregation chains "from the middle" without knowing the previous message sequence. To capture those attacks we modify the aggregation oracle such that it returns aggregates for sequences of messages, starting now with an arbitrary aggregate-so-far. Thus, we also incorporate some ideas of the *aggregation-unforgeability* notion [10] into our new model.

Aggregation-unforgeability here demands that the adversary cannot output a valid chain, unless its a trivial combination of previous aggregation queries and values by corrupt parties. An example of such a trivial combination is depicted in Figure 1, where the adversary computes the final value by simply iterating through the sequence with the help of the aggregation oracle and local computations by corrupt players. Note that each aggregation query is for a sequence of honest parties and this requires several public keys.

*Attack Scenario.* As in the aggregation-unforgeability model of Eikemeier et al. for aggregated MACs, we also grant the adversary in our model an aggregation oracle returning aggregates for (ordered) *sets* of messages. To allow reasonable aggregation queries we hand the adversary now $t$ genuine public keys $pk_1, \ldots, pk_t$ of initially honest parties as in [15], instead of considering a single challenge key as in the chosen-key model [4,6].

The adversary's attack is divided into two phases. In the first phase, the adversary has access to a corruption and a key-setting oracle, both initialized with the $t$ key pairs $((sk_1, pk_1) \ldots, (sk_t, pk_t))$. By querying the corruption oracle the adversary can obtain at most $t-1$ secret keys of his choice. We denote by $Q_{\mathsf{Cor}}$ the set of corrupted keys. To model rogue-key attacks we also provide an oracle $\mathsf{SetKey}$ which allows the adversary to change the public key of a previously corrupted party, i.e., on input $pk, pk^*$ the oracle replaces the public key $pk$ of a corrupt party by $pk^*$. Recall that we assume that public keys must be unique. Any modifications of corrupted keys are captured by the set $Q_{\mathsf{Cor}}$ as well.

The adversary starts the second phase by interacting with the sequential aggregate signature oracle $\mathsf{OSeqAgg}$ but is denied access to the corruption or key-setting oracle in this phase (reflecting static corruptions[3]). On input of an aggregate-so-far $\sigma'$, a sequence of new messages $\mathbf{M}$ for public keys $\mathbf{pk}$ the $\mathsf{OSeqAgg}$ oracle checks whether all public keys in $\mathbf{pk}$ are distinct and belong to honest parties. If an invalid public key appears $\mathsf{OSeqAgg}$ answers $\perp$, otherwise it responds with a new valid aggregate $\sigma$ derived by running the aggregation algorithm stepwise for all input data. We remark that the aggregation oracle

[3] We observe that the standard strategy to lift security against static corruptions to security against adaptive corruptions by guessing the right "target" key in advance does not work in our setting, as our security notion relies on multiple honest users.

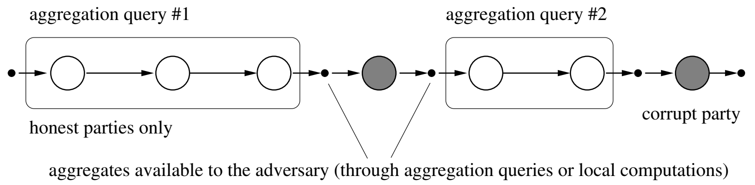
aggregation query #1
aggregation query #2
honest parties only
corrupt party
aggregates available to the adversary (through aggregation queries or local computations)

**Fig. 1.** Example of a trivial combination of replies to aggregation queries and local computations by corrupt parties

only aggregates for honest parties, i.e., where the corresponding keys were neither corrupted nor modified; for corrupt players the adversary, holding the secret key, must add the values herself.

Eventually the adversary $\mathcal{A}$ halts, outputting a tuple $(\mathbf{pk}^*, \mathbf{M}^*, \sigma^*)$. The forgery must be valid according to our definition of history-free sequential aggregate signature schemes. In addition, the signature must be non-trivial which is quantified by defining the closure of all query/answer pairs of $\mathcal{A}$. Here, we denote by $Q_{\mathsf{Seq}}$ the set of all query/answer tuples $((\sigma', \mathbf{M}, \mathbf{pk}),\ \sigma)$ that occur in $\mathcal{A}$'s interaction with the $\mathsf{OSeqAgg}$ oracle. Recall that $Q_{\mathsf{Cor}}$ denote the sets of all keys that were corrupted and possibly modified by the adversary. The closure contains all admissible combinations of aggregated data for the queried sequences together with all possible values by corrupted parties.

*Closure.* For history-free sequential aggregate signatures, defining the closure is more complex as in the general case that we discuss in Section 5. Here, an adversary can query partial chains and later possibly combine several of them by using corrupted keys or chains with matching starting/end points. Thus, we define the closure recursively through a function $\mathsf{Trivial}_{Q_{\mathsf{Seq}},Q_{\mathsf{Cor}}}$ which, for parameters $(\mathbf{pk}, \mathbf{M}, \sigma)$ describes all sequences that can be derived trivially starting from message sequence $\mathbf{M}$ and aggregate-so-far $\sigma$, i.e., where one can append (recursively expanded) trivial sequences via aggregation queries or local computations by corrupt players. For example, if we have an aggregation query $(\sigma_0, \mathbf{pk}, \mathbf{M})$ with answer $\sigma$ in $Q_{\mathsf{Seq}}$ and another query $(\sigma, \mathbf{pk}', \mathbf{M}')$ with the answer from the first query as the starting aggregate, then the sequence $(\mathbf{pk}||\mathbf{pk}', \mathbf{M}||\mathbf{M}')$ is in the trivial set. So is any extension of this sequence for corrupt players. We note that, if the final aggregate of a chain and the starting aggregate do not match, then the combined sequence is not in the closure, neither are subsequences of previous queries (unless either sequence appears in another query).

The closure is then defined to contain all trivial sequences starting from the information available to the adversary at the beginning, namely, the empty message sequence, the starting key $pk_0 = \emptyset$ and the starting tag $\sigma_0 = \emptyset$. Note that the closure here is now a set of tuples where each tuple represents a sequential aggregation.

**Definition 4 (Sequential Closure of $\mathcal{A}$'s queries).** *Let $Q_{\mathsf{Cor}}$ and $Q_{\mathsf{Seq}}$ be the sets corresponding to the different oracle responses and let $\mathsf{Trivial}_{Q_{\mathsf{Seq}},Q_{\mathsf{Cor}}}$ be a recursive function of trivial combinations defined as*

$$\begin{aligned}\mathsf{Trivial}_{Q_{\mathsf{Seq}},Q_{\mathsf{Cor}}}(\mathbf{pk},\mathbf{M},\sigma) \\ := \quad \{(\mathbf{pk},\mathbf{M})\} \cup \bigcup_{((\sigma,\overline{\mathbf{M}},\overline{\mathbf{pk}}),\overline{\sigma})\in Q_{\mathsf{Seq}}} \mathsf{Trivial}_{Q_{\mathsf{Seq}},Q_{\mathsf{Cor}}}(\mathbf{pk}||\overline{\mathbf{pk}},\mathbf{M}||\overline{\mathbf{M}},\overline{\sigma}) \\ \cup \bigcup_{\substack{\forall \overline{M},\overline{\sigma} \\ \wedge pk_i \in Q_{\mathsf{Cor}}}} \mathsf{Trivial}_{Q_{\mathsf{Seq}},Q_{\mathsf{Cor}}}(\mathbf{pk}||pk_i,\mathbf{M}||\overline{M},\overline{\sigma})\,.\end{aligned}$$

*The closure* Closure *of $\mathcal{A}$'s queries $Q_{\mathsf{Seq}}$ and $Q_{\mathsf{Cor}}$ is then defined by recursively generating the trivial combinations starting from the empty tuple as described above:*

$$\mathsf{Closure}(Q_{\mathsf{Seq}},Q_{\mathsf{Cor}}) := \mathsf{Trivial}_{Q_{\mathsf{Seq}},Q_{\mathsf{Cor}}}(\emptyset,\emptyset,\emptyset).$$

As an example consider an attack on a regular (non-aggregate) signature scheme, with a single honest party and no corrupt players. Then the closure contains all queries to the signing oracle and renders these values as trivial. Note that we do not treat the case of concatenating answers for the same public key in any special way.

A more important example are the mix-and-match attacks in which the adversary sees several aggregation chains (of honest parties) but is able to combine them into a new sequence. This new sequence would then be not in the closure and thus constitute a legitimate forgery attempt. In other words, any secure scheme according to our notion must prevent such mix-and-match attacks.

*Aggregation Unforgeability.* With the definition of the sequential closure, we propose the following security model for history-free sequential aggregate signatures.

**Definition 5 (Aggregation Unforgeability).** *A history-free sequential aggregate signature scheme* $\mathsf{SAS} = (\mathsf{SeqKg},\mathsf{SeqAgg},\mathsf{SeqAggVf})$ *is* aggregation-unforgeable *if for any efficient algorithm $\mathcal{A}$ (working in modes* CORRUPT, FORGE*) the probability that the experiment* $\mathsf{SeqForge}^{\mathsf{SAS}}_{\mathcal{A}}$ *evaluates to* 1 *is negligible (as a function of n), where*

```
***Experiment*** $\mathsf{SeqForge}^{\mathsf{SAS}}_{\mathcal{A}}(n)$
    $(sk_1,pk_1),\ldots,(sk_t,pk_t) \leftarrow \mathsf{SeqKg}(1^n)$
    $\mathbf{K}' \leftarrow ((sk_1,pk_1),\ldots,(sk_t,pk_t))$
    $\mathsf{st} \leftarrow \mathcal{A}^{\mathsf{Corrupt}(\mathbf{K}',\cdot),\mathsf{SetKey}(\mathbf{K}',\cdot,\cdot)}(\text{CORRUPT},pk_1,\ldots.pk_t)$
        // it is understood that $\mathcal{A}$ keeps state st
    Let $\mathbf{K}$ be the set of the updated keys of all parties
    $(\mathbf{pk}^*,\mathbf{M}^*,\sigma^*) \leftarrow \mathcal{A}^{\mathsf{OSeqAgg}(\mathbf{K},\cdots)}(\text{FORGE},\mathsf{st})$
    Return 1 iff $pk_i \neq pk_j$ for all $i \neq j$ and $\mathsf{SeqAggVf}(\mathbf{pk}^*,\mathbf{M}^*,\sigma^*) = 1$ and
        $(\mathbf{pk}^*,\mathbf{M}^*) \notin \mathsf{Closure}(Q_{\mathsf{Seq}},Q_{\mathsf{Cor}})$.
```

*Relaxed Security Notion.* Our definition is very demanding in the sense that prefixes of aggregation sequences are considered to be non-trivial. In particular, this means that intermediate values in such a chain cannot be available to the

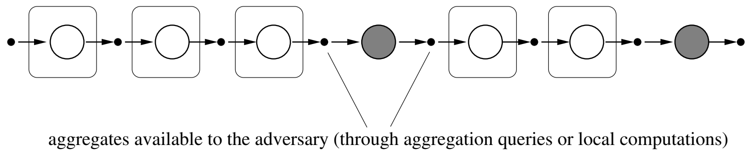
aggregates available to the adversary (through aggregation queries or local computations)

**Fig. 2.** Relaxed Security Notion: In comparison to the stronger notion (Figure 1) the adversary can only make aggregation queries of length 1. The closure potentially allows more combinations now and thus rules out more sequences as trivial.

adversary, or else successful attacks according to our model are straightforward. This model corresponds to the case that the forwarded data between honest parties are for instance encrypted.

Regarding existing sequential aggregate signature schemes like [14], all intermediate signatures that appeared in the computation of the final aggregate can be re-obtained by simply verifying the aggregate signature, since the verification algorithm "peels off" the aggregate. Thus, we also propose a relaxed definition of history-free unforgeability that takes the possibility of obtaining the *intermediate* signatures into account, inciting the name *mezzo aggregation unforgeability.*

We also remark that a simple approach like having the first party in a sequence create some unique identifier or nonce, which is used by all subsequent players, usually does not facilitate the design of schemes because the adversary can always put a corrupt player upfront. Similarly to the case of non-ordered aggregation we can have a solution with counters or time stamps but this again requires synchronization between the parties.

We can easily cast the weaker notion in our model by allowing only aggregation queries for sequences of length one, i.e., where the adversary has to compute longer chains itself by iterating through the sequence manually. Clearly, this adversary is a special case of our adversary above and the security guarantee is therefore weaker (in other words, the closure now contains more trivial elements). It is also very easy to prove this formally by considering a scheme where the new aggregate contains the previous aggregate. For the stronger notion this allows to obtain a valid aggregate of a prefix easily, whereas for the weaker notion the extra aggregate is already been input by the adversary and thus provides no additional information. The difference between the models is depicted in Figure 2.

**Definition 6 (Mezzo Aggregation Unforgeability).** *A history-free sequential aggregate signature scheme* $\mathsf{SAS} = (\mathsf{SeqKg}, \mathsf{SeqAgg}, \mathsf{SeqAggVf})$ *is* mezzo aggregation-unforgeable *if it is aggregation-unforgeable for any efficient algorithm* $\mathcal{A}$ *that only calls oracle* $\mathsf{OSeqAgg}$ *for sequences of length one.*

We note that mix-and-match attacks are still ruled out by the above definition. For this observe that any "manually iterated" sequence can only interfere with other sequences if intermediate signatures collide. Such collisions are, however,

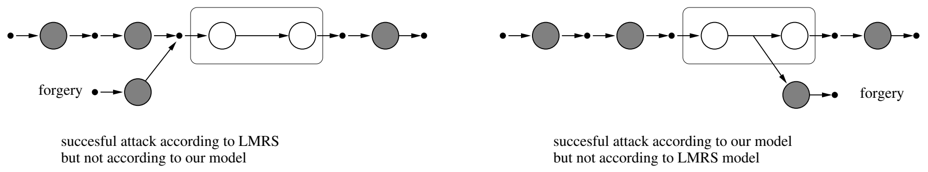
forgery
succesful attack according to LMRS
but not according to our model
forgery
succesful attack according to our model
but not according to LMRS model

**Fig. 3.** Comparison of the LMRS security model and our (strong) model: Prepending any values by corrupt parties is not considered a successful attack in our model (left part), whereas branching into a different sequence from some intermediate value is not considered a successful attack in the LMRS model (right part)

unlikely and can only happen with negligible probability. Else, such collisions would easily allow to forge individual signatures of honest parties and would constitute a successful forgery in the above sense.

## 3.3 Relationship to the LMRS-Model

It is easy to see that our security model is strictly stronger than the one by Lu et al. [13] because successful attacks according to their definition involve individual forgeries for fresh messages against a single challenge key (which thus cannot belong to our closure). At the same time their approach does not allow to verify the order of aggregation steps, whereas changing the order constitutes a successful attack according to our definition. We therefore focus on the comparison to the LMRS-model.

On one hand our model gives the adversary more power than in the LMRS-model for secure sequential aggregation, because it does not need to specify the starting message sequence for aggregation queries. On the other hand we allow the adversary less freedom when it comes to values of corrupt players in the forgery attempt. Hence, the possibilities in the attack are somewhat compensated for and this makes the models incomparable, as we show by the following separating examples.

The ideas of the separating examples are given in Figure 3. The left part of the figure shows an attack which is defined as trivial in our model but constitutes a break in the LMRS model. Indeed, it seems that in the history-free setting the adversary can always find "bad" keys for corrupt parties which enable collisions on the intermediate values. Since the information about the starting sequence then does not enter the further computations preventing such attacks in our setting seems impossible. The right side shows a successful attack in our model which takes advantage of a prefix of an aggregation subsequence; this is by definition not a successful attack in the LMRS model. A similar separation holds for our relaxed notion. We discuss these cases in more detail in the full version.

We briefly discuss how to add the LMRS security property to our (mezzo) aggregation unforgeability, if now all preceding public keys **pk** and messages **M** in a sequence are known. The idea is to use a collision-resistant hash function

$h$ and, for each signature creation, to prepend the hash value $c = h(\mathbf{pk}, \mathbf{M})$ to the message to be signed. For verification one does the same. Note that, while this is formally not a history-free scheme anymore, signing still does not require verification of preceding signatures.

Aggregation unforgeability still holds in the modified scheme if we consider each hash value to be an integral part of the message to be signed. But the collision resistance of the hash function $h$ now also ensures $\mathsf{LMRS}$ security, because we can assume that all hash values of sequences are unique. This implies that in an $\mathsf{LMRS}$ forgery attempt the message with the prepended hash value has not been signed by the honest user in question (either the prefix is new and thus the hash value, or the message in combination with this sequence is). This means that the forgery sequence is not in the closure and would thus constitute a breach of (mezzo) aggregation unforgeability.

## 4 Construction

We derive a history-free sequential aggregate signature scheme based on the $\mathsf{BLS}$ signature scheme that is secure in the random oracle model [7]. This scheme has already been successfully applied to derive the non-sequential BGLS aggregate signature scheme [6]. Below we assume that we have an efficient, non-degenerate bilinear map $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3$ for system-wide available groups, where $g_1$ is a generator of $\mathbb{G}_1$ and $g_2$ is a generator of $\mathbb{G}_2$. We assume that $\mathbf{e}(\cdot, g_2)$ is one-to-one. Also, let $H : \{0,1\}^* \mapsto \mathbb{G}_1$ be a public hash function.

In the $\mathsf{BLS}$ signature scheme the key generation algorithm $\mathsf{Kg}(1^n)$ picks an element $x \leftarrow \mathbb{Z}_p$ at random and computes $v \leftarrow g_2^x$. It returns $(pk, sk) \leftarrow (v, x)$. The signing algorithm $\mathsf{Sign}(x, M)$ takes as input a message $M \in \{0,1\}^*$ and a secret key $x$. It computes $\sigma \leftarrow H(M)^x$ and returns the signature $\sigma \in \mathbb{G}_1$. The verification algorithm $\mathsf{Vf}(v, M, \sigma)$ outputs 1 iff $\mathbf{e}(\sigma, g_2) = \mathbf{e}(H(M), v)$.

### 4.1 Construction Based on BLS Signatures

The idea of our construction is as follows. We let the signer build a link between all previous all signatures by linking them through a hash chain. That is, in each aggregation step the signer receives the aggregate-so-far $(\sigma', pk', c', s')$, consisting of an aggregate $\sigma'$, the public key $pk'$ of the preceding signer, a hash chain value $c'$ and the non-aggregated signature $s'$ of the preceding party. The signer first checks that $s'$ is a valid signature under $pk'$ for $c'$ and, if so, it extends the hash chain via $c \leftarrow h(\mathbf{e}(\sigma', g_2), M, pk', c')$ for its message $M$. Note that using the value under the bilinear mapping instead of $\sigma'$ is necessary for the verification the whole sequence without knowing the individual aggregates and is a specific property of the $\mathsf{BLS}$ scheme. The signer next computes a non-aggregated signature $s$ for $c$ and aggregates $s$ to $\sigma'$ to derive $\sigma$, and finally forwards $(\sigma, pk, c, s)$ to the next signer.

**Construction 1.** *Let* $\mathsf{DS} = (\mathsf{Kg}, \mathsf{Sig}, \mathsf{Vf})$ *be the* $\mathsf{BLS}$ *signature scheme and* $h : \{0,1\}^* \mapsto \{0,1\}^n$ *be a hash function. Define the following efficient algorithms:*

**Key Generation.** *The key generation algorithm is identical to* $\mathsf{Kg}$.

**Sequential Signature Aggregation.** *Algorithm* $\mathsf{SeqAgg}$ *gets as input a pair of keys* $(sk, pk) = (x, v)$*, a message* $M \in \{0,1\}^*$*, and a sequential aggregate signature* $(\sigma', pk', c', s')$*. The algorithm sets* $c \leftarrow h(\mathbf{e}(\sigma', g_2), M, pk', c')$*, where* $\mathbf{e}(\emptyset, g_2) = 1$ *by definition, checks that* $\mathsf{Vf}(pk', c', s') = 1$ *or that* $pk', c', s' = \emptyset$ *are the starting symbols, and stops if not. Else it computes the signature* $s = H(c)^x \leftarrow \mathsf{Sig}(sk, c)$ *on* $c$ *and the value* $\sigma \leftarrow \sigma' \cdot s$*. It outputs the sequential aggregate signature* $(\sigma, pk, c, s)$*.*

**Aggregate Verification.** *The input of algorithm* $\mathsf{SeqAggVf}(\mathbf{pk}, \mathbf{M}, \sigma)$ *is a sequence of public keys* $\mathbf{pk} = (pk_1, \ldots, pk_\ell)$*, a sequence of messages* $\mathbf{M} = (M_1, \ldots, M_\ell)$ *as well as an aggregate* $\sigma$ *(with* $pk, c, s$*). It parses* $pk_i = g_2^{x_i}$*, sets*

$$c_0 \leftarrow \emptyset \quad \text{and} \quad pk_0 \leftarrow \emptyset \quad \text{and} \quad c_i \leftarrow h\Big(\prod_{j=0}^{i-1} \mathbf{e}(H(c_j), pk_j), M_i, pk_{i-1}, c_{i-1}\Big)$$

*for* $i = 1, \ldots, \ell$*, where* $\mathbf{e}(H(\emptyset), pk_j) = 1$ *by definition, and outputs* $1$ *if*

$$\mathbf{e}(\sigma, g_2) = \prod_{i=1}^{\ell} \mathbf{e}(H(c_i), g_2^{x_i}).$$

Completeness follows inductively, as for honest parties each intermediate aggregate $\sigma_i$ is a valid signature for $c_1, \ldots, c_i$ and therefore the next output also satisfies $\mathbf{e}(\sigma_i, g_2) = \mathbf{e}(H(c_j), g_2^{x_i})$.

## 4.2 Security

Our security proof basically follows by reduction to the security to the BLS signature scheme and the collision resistance of $h$. We note that we do not explicitly rely on the random oracle model, only implicitly through the (currently best) security proof for the BLS scheme. Instead, we could give a straight reduction to the co-Diffie-Hellman problem [7], but then we would need to program the random oracle. The main idea of the proof is that we either break the underlying $\mathsf{BLS}$ scheme (in case $\mathbf{C}^*$ computed in the verification of the adversary's forgery attempt contains a new value $c_i^*$), or that the adversary has to forge a (regular) signature for an honest party or to find a collision for $h$ (if all values in $\mathbf{C}^*$ have appeared during the attack).

**Theorem 2.** *Let* $h$ *be a collision-resistant hash function. If the* $\mathsf{BLS}$ *signature scheme is unforgeable, then the scheme defined in Construction 1 is a history-free, mezzo aggregation-unforgeable sequential aggregate signature scheme.*

*Proof.* We prove this theorem assuming towards contradiction that there exists an adversary $\mathcal{A}$ breaking aggregation-unforgeability with non-negligible probability $\epsilon(n)$. Assume that this adversary eventually outputs a valid forgery $\mathbf{M}^*$, $pk^*$ and $\sigma^*$. Let $\mathbf{C}^* = (c_1^*, \ldots, c_\ell^*)$ denote the values derived during the verification, and assume that the sequence $\mathbf{M}^*$ does not belong to the closure.

If the probability that the adversary $\mathcal{A}$ succeeds and there is some $c_i^*$ for an honest party which has never been queried to an aggregation query for this party, then we can break the underlying aggregate signature scheme. To this end we construct an algorithm $\mathcal{B}$ (receiving a challenge key and having access to a signature oracle for this key) as follows:

**Setup.** Algorithm $\mathcal{B}$ gets as input a public key $pk_c$, it picks $t-1$ key pairs $(sk_i, pk_i) \leftarrow \mathsf{SeqKg}(1^n)$ and inserts the key $pk_c$ at a random position, $\mathbf{pk} \leftarrow (pk_1, \ldots, pk_{j-1}, pk_c, pk_{j+1} \ldots, pk_t)$. $\mathcal{B}$ simulates $\mathcal{A}$ in a black-box way on input $\mathbf{pk}$ (if we assume $H$ to be a random oracle then $\mathcal{B}$ grants $\mathcal{A}$ direct access to $H$).

**Key Oracles.** During the simulation, $\mathcal{A}$ is allowed to corrupt keys and to change them. If $\mathcal{A}$ invokes the corruption oracle $\mathsf{Corrupt}(\mathbf{sk}, \cdot)$ on input $pk$, then $\mathcal{B}$ returns $sk_i$ if $pk_i = pk$, for some $i \in \{1, \ldots, t\} \setminus j$, and otherwise $\mathsf{failed}$. In the case that $\mathcal{A}$ wishes to substitute a certain public key $pk \in \mathbf{pk}$ and queries its key-modification oracle $\mathsf{SetKey}(\mathbf{sk}, \cdot)$ about a pair $(pk, pk')$, then $\mathcal{B}$ sets $pk_i = pk'$ if $pk_i = pk$ for an index $i \in \{1, \ldots, t\} \setminus j$. It returns $\mathsf{succ}$ if such a public key exists and substitution succeeded, otherwise $\mathsf{failed}$.

**Aggregate Signing.** Whenever $\mathcal{A}$ asks the aggregate signing oracle $\mathsf{SeqAgg}$ to build a new sequential aggregate signature for an aggregate-so-far $\sigma'$, a message $M$, and a public key $pk$, algorithm $\mathcal{B}$ answers this query in the following way. It first checks if the public key $pk$ has never been corrupted nor substituted (if so, it returns $\perp$). Adversary $\mathcal{B}$ either computes the aggregate invoking its external signing oracle (in the case where $pk = pk_c$), or else by executing the signing algorithm itself (for the corresponding secret key $sk$). In both cases all other steps of the aggregation algorithm besides the signing step can be computed easily. $\mathcal{B}$ outputs the full aggregate to $\mathcal{A}$.

**Output.** At the end of the simulation $\mathcal{A}$ outputs a tuple $(\mathbf{M}^*, \mathbf{pk}^*, \sigma^*)$. Algorithm $\mathcal{B}$ computes $\mathbf{C}^*$ as in the description of the verification procedure and returns these values together with $\mathbf{pk}^*$ and $\sigma^*$. Algorithm $\mathcal{B}$ checks if $pk_c \in \mathbf{pk}^*$ and, if so, computes the values $c_i^*$ as for verification, and outputs $c_c^*$ together with $\sigma^* \cdot \prod_{i \neq c} H(c_i)^{-x_i}$ as the signature (for the known secret keys $x_i$ belonging to the other parties in $\mathbf{pk}^*$).

For the analysis note that, in the case that some new $c_i^*$ for some honest party is in $\mathbf{C}^*$ our algorithm $\mathcal{B}$ loses only a factor $1/t$ for guessing the right public key. But then, for a valid forgery of $\mathcal{A}$ we have $\mathbf{e}(\sigma^*, g_2) = \prod_{i=1}^{\ell} \mathbf{e}(H(c_i^*), g_2^{x_i})$. Dividing out $\prod_{i \neq c} H(c_i^*)^{x_i}$ of $\sigma^*$ yields $\mathbf{e}(\sigma^* \cdot \prod_{i \neq c} H(c_i^*)^{-x_i}, g_2) = \mathbf{e}(H(c_c^*), pk_c)$ and therefore a valid forgery for the BLS scheme under public key $pk_c$. Hence, this case of $\mathcal{A}$ winning cannot have non-negligible probability.

Next assume that all the $c_i^*$'s of honest parties have appeared in aggregation requests before (and are answered without failure), but $\mathcal{A}$ still wins. In the forgery attempt consider the leftmost honest party at position $i$ such that the leading sequence $(M_1^*, \ldots, M_i^*)$ of $\mathbf{M}^*$ does not lie in the closure. Since we assume that $c_i^*$ has appeared in some aggregation query to party $i$ before, we must have a query $(\sigma', pk', c'), M$ with

$$h(\mathbf{e}(\sigma', g_2), M, pk', c') = c_i^* = h(\prod_{j<i} \mathbf{e}(H(c_j^*), pk_j), M_i^*, pk_{i-1}, c_{i-1}).$$

By the collision-resistance of $h$ we conclude that $M = M_i^*$, $pk' = pk_{i-1}$ and $c' = c_{i-1}^*$ and $\mathbf{e}(\sigma', g_2) = \prod_{j<i} \mathbf{e}(H(c_j^*), pk_j)$. By assumption, the leading sequence $(M_1^*, \ldots, M_i^*)$ is not in the closure. There are three cases:

- Our "target" party at position $i$ is the first one in the sequence $(M_1^*, \ldots, M_i^*)$, i.e., $i = 1$. Since it then only computes an aggregate if $\sigma', pk', c' = \emptyset$ we derive the contradiction that the sequence is in fact in the closure, due to the aggregation query $(\sigma', pk', c'), M = M_i^*$ yielding $c_i^*$. This, however, contradicts our assumption.
- Assume that there is a corrupt party at position $i-1$ in the forgery sequence. Then, by construction and since party $i$ is the leftmost with the sequence $(M_1^*, \ldots, M_i^*)$ not being in the closure, the sequence including the corrupt party must be in the closure (all subsequences must already be in the closure by assumption). But then the query triggering the appearance of $c_i^*$ again makes $(M_1^*, \ldots, M_i^*)$ per definition also part of the closure. This is so since corrupt parties can "link" any trivial sequences.
- The final case is if there is an honest party at position $i-1$. Note that our party at $i$ only returns an aggregate if the signature $s'$ is a valid signature for the incoming value $c' = c_i^*$ under the same key $pk' = pk_{i-1}$ of the honest party at position $i-1$. We conclude again that the adversary needs to make the honest party at some step sign $c_i^*$ (or needs to forge a signature for honest party at $i-1$, which would again contradict the security of the BLS signature scheme). However, by the collision resistance of $h$ and noting that the function $\mathbf{e}(\cdot, g_2)$ is one-to-one, it follows that this requires the same input $(\sigma'', pk'', c''), M'$ to the party at position $i-1$ as on the "closure path". Furthermore, the valid signatures $s''$ for $c''$ and $s'$ for $c'$ are unique, and it therefore follows again that the closure extends to the party at position $i$, contradicting again our assumption.

This shows that the adversary can win in this case with negligible probability only, and concludes the proof. □

We again note that, following the discussion in Section 3.3, we can easily get a (non history-free) signature scheme which is simultaneously also LMRS secure without the verification of signatures in the sequence, This is achieved by inserting the sequence of messages and public keys into the evaluation of the hash function $h$.

## 5 Security of Non-sequential Aggregation Schemes

The common security model for non-sequential aggregate signatures of Boneh et al. [6] only considers limited attacks (akin to our weaker security notion), even though stronger notions may be desirable for some applications (similar to our strong notion). For the case of symmetric authentication this was already discussed in [10] by presenting an attack against an aggregate MAC scheme, that was outside of the previous security model. Here we show that a similar argumentation holds for aggregate signatures as well.

*Mix-and-Match Attacks.* We first recall the example of an "mix-and-match" attack that was given for aggregate MACs by Eikemeier et al. From an abstract point of view, the attack uses three aggregates for message sets $\{M_1, M_2\}$, $\{M_3, M_2\}$ and $\{M_1, M_4\}$ to derive a valid aggregate for a fourth pair $\{M_3, M_4\}$. The attack is not considered a security breach according to the model by [6]. Roughly, the shortcoming is due to the definition of "trivial" attacks: an adversary is usually not considered to succeed if the messages in the forgery have been authenticated *individually* during the attack. In the example above this means that *any* combination of the messages $M_1, M_2, M_3, M_4$ cannot be used for a successful forgery, although only three of these combinations have actually appeared before. Ideally, however, an aggregation scheme should be considered insecure if an adversary is able to transform several aggregates into a *new combination* that has not been authenticated before.

More concretely, recall that an aggregate in the scheme by Boneh et al. is of the form $\sigma = \prod \sigma_i$ for regular BLS signatures $\sigma_i = H(M_i)^{x_i}$ for random oracle $H$, message $M_i$ and secret key $x_i$. The public key is given by $g^{x_i}$ and verification is performed with the help of the pairing operation. Given the replies

$$\sigma_1 = H(M_1)^{x_1} \cdot H(M_2)^{x_2}, \quad \sigma_2 = H(M_3)^{x_1} \cdot H(M_2)^{x_2}, \quad \sigma_3 = H(M_1)^{x_1} \cdot H(M_4)^{x_2}$$

to three aggregation queries for message sets $\{M_1, M_2\}$, $\{M_3, M_2\}$ and $\{M_1, M_4\}$, the adversary is able to compute a valid aggregate

$$\sigma^* = \sigma_1^{-1} \cdot \sigma_2 \cdot \sigma_3 = H(M_3)^{x_1} \cdot H(M_4)^{x_2}$$

for the set $\{M_3, M_4\}$. According to the definition of [6] this, however, does not constitute a security breach.

*Relation to Boneh's et al. Aggregate Extraction Problem.* Our mix-and-match attack on the scheme of Boneh et al. [6] benefits from the fact that we can remove some signatures from the aggregate. Interestingly, the authors in [6] already address the question whether it is possible to extract any subset of (unknown) signatures from the aggregate or not. This problem is called aggregate-extraction-problem. Extracting even a single (unknown) signature from such an aggregate is equivalent to solving the computational Diffie-Hellman problem, as subsequently shown by Coron and Naccache [9]. Thus, in a sense, our result can also be seen as a generalization of the aggregate extraction problem with respect to the BGLS aggregate signature scheme, to a more general context where we not only consider the extraction of single signatures, but also the *(re-)combination* of aggregates (as discussed above).

*Defining Stronger Aggregation Unforgeability.* To derive a stronger security notion Eikemeier et al. adapt their notion and attack model for the sequential case, except that the aggregation oracle now takes *unordered* sets of messages and public keys. The definition of the closure for our signature case simplifies and is then given by

$$\mathsf{Closure}(Q_{\mathsf{Agg}}, Q_{\mathsf{Cor}}) = \left\{ \bigcup_{M_A \in A} M_A \cup M_C \;\middle|\; A \subseteq Q_{\mathsf{Agg}},\, M_C \subseteq \bigcup_{pk^* \in Q_{\mathsf{Cor}}} \{(pk^*, M) \,|\, M \in \{0,1\}^*\} \right\}.$$

We remark again that it is unknown whether this notion can indeed be satisfied.

*Synchronized Aggregate Signatures.* A line of research studies aggregate signatures where signers share a synchronized clock [11,2,1], showing that efficient constructions under well known computational assumptions are possible in this model, even for unordered aggregation. Following this line, Eikemeier et al. [10] discuss how to derive MAC schemes secure according to a relaxed notion similar to the one above, and their ideas transfer to signatures as well. However, their solution still does not cover deletion attacks. Furthermore, it is of course preferable to avoid such synchronization assumptions.

**Acknowledgments.** We thank Leo Reyzin for discussions and clarifications about [8], especially about the (un)suitability of different security notions for the secure routing problem.

This work was supported by the Emmy Noether Program Fi 940/2-1 and the Heisenberg Program Fi 940/3-1 of the German Research Foundation (DFG) and by CASED (`www.cased.de`). This work was partially supported by the US Army Research Laboratory and the UK Ministry of Defense 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 Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation herein. This work was also supported by the German Ministry for Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA — `www.cispa-security.de`). Part of the work of the second and third author done while being at Darmstadt University.

## References

1. Ahn, J.H., Green, M., Hohenberger, S.: Synchronized aggregate signatures: New definitions, constructions and applications. In: Annual Conference on Computer and Communications Security (CCS), pp. 473–484. ACM Press (2010)
2. Bagherzandi, A., Jarecki, S.: Identity-Based Aggregate and Multi-Signature Schemes Based on RSA. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 480–498. Springer, Heidelberg (2010)
3. 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)
4. 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)
5. Boldyreva, A., Gentry, C., O'Neill, A., Yum, D.H.: New multiparty signature schemes for network routing applications. ACM Trans. Inf. Syst. Secur. 12(1) (2008)
6. 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)
7. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. Journal of Cryptology 17(4), 297–319 (2004)
8. Brogle, K., Goldberg, S., Reyzin, L.: Sequential aggregate signatures with lazy verification. Cryptology ePrint Archive: Report 2011/222 (2011), `http://eprint.iacr.org/2011/222l`
9. Coron, J.-S., Naccache, D.: Boneh *et al.*'s $k$-Element Aggregate Extraction Assumption Is Equivalent to the Diffie-Hellman Assumption. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 392–397. Springer, Heidelberg (2003)
10. 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)
11. Gentry, C., Ramzan, Z.: Identity-Based Aggregate Signatures. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 257–273. Springer, Heidelberg (2006)
12. Lepinski, M.: BGPSec protocol specification. IETF Internet-Draft (2011)
13. 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)
14. 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)
15. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures: extended abstract. In: Annual Conference on Computer and Communications Security (CCS), pp. 245–254. ACM Press (2001)
16. Neven, G.: Efficient Sequential Aggregate Signed Data. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 52–69. Springer, Heidelberg (2008)
17. Schröder, D.: How to Aggregate the CL Signature Scheme. In: Atluri, V., Diaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 298–314. Springer, Heidelberg (2011)