History-Free Aggregate Message Authentication Codes

Oliver Eikemeier[1], Marc Fischlin[1], Jens-Fabian Götzmann[1], Anja Lehmann[1,2], Dominique Schröder[1], Peter Schröder[1], and Daniel Wagner[1]

[1] Darmstadt University of Technology, Germany — www.minicrypt.de
[2] IBM Research Zurich, Switzerland

Abstract. Aggregate message authentication codes, as introduced by Katz and Lindell (CT-RSA 2008), combine several MACs into a single value, which has roughly the same size as an ordinary MAC. These schemes reduce the communication overhead significantly and are therefore a promising approach to achieve authenticated communication in mobile ad-hoc networks, where communication is prohibitively expensive. Here we revisit the unforgeability notion for aggregate MACs and discuss that the definition does not prevent "mix-and-match" attacks in which the adversary turns several aggregates into a "fresh" combination, i.e., into a valid aggregate on a sequence of messages which the attacker has not requested before. In particular, we show concrete attacks on the previous scheme.

To capture the broader class of combination attacks, we provide a stronger security notion of aggregation unforgeability. While we can provide stateful transformations lifting (non-ordered) schemes to meet our stronger security notion, for the statefree case we switch to the new notion of history-free sequential aggregation. This notion is somewhat between non-ordered and sequential schemes and basically says that the aggregation algorithm is carried out in a sequential order but must not depend on the preceding messages in the sequence, but only on the shorter input aggregate and the local message. We finally show that we can build an aggregation-unforgeable, history-free sequential MAC scheme based on general assumptions.

1 Introduction

Aggregate message authentication codes [5] allow the aggregation of multiple MACs, generated by different senders for possibly different messages, such that the aggregate has the same size as a single MAC. These MACs are especially suited for settings involving resource-constrained devices like mobile ad-hoc networks (MANET). Thereby, the communication is very power-consuming and asymmetric primitives like signatures are prohibitively expensive due to the limited computational power of the devices. In this case, verification of an aggregated tag can be carried out by any receiver that shares all secret keys with the participating senders, e.g., a base station collecting data from the sensors.

Security Revisited. The unforgeability notions for aggregate MACs follow the known principle for aggregate signature schemes [2]. Basically, it states that an adversary, who controls all aggregating parties except for a single signer, cannot create a valid aggregate for a "fresh" set of messages. Here a set is considered fresh if the designated signer is in this set but has not signed the message before. In other words, the

unforgeability of the aggregation scheme is tied to the unforgeability of individual messages.

Aggregation, however, is about combining data, and protection of individual messages may not be sufficient in all settings: deleting parts, re-ordering entries, extending or recombining aggregates to a new valid aggregate may be serious threats for applications. We illustrate our idea with a simple (sequential) network of nodes

$$N_1 \longrightarrow N_2 \longrightarrow N_3 \longrightarrow N_4.$$

The aggregation scheme should be used to authenticate routing paths, where N_i only accepts input from N_{i-1}, augments the aggregate-so-far by a MAC of its own identity before forwarding the new aggregate to N_{i+1}. Then, if one is able to delete for example N_2's contribution from the aggregate, one obtains a valid authentication of an invalid route $N_1 \rightarrow N_3 \rightarrow N_4$. According to the definition of [5], however, the above attack does not constitute a security breach, as no individual MAC has been forged. We discuss similar, concrete attacks of this "mix-and-match" type on the aggregate MAC scheme due to Katz and Lindell [5] in Appendix A.

Aggregation Unforgeability. To cover advanced attacks as discussed above, we introduce our stronger security notion of *aggregation unforgeability*. The attacker's mode of operation is similar to the definitions of [5], i.e., it can make aggregation queries for messages of its choice and should eventually produce a forgery. However, in order to capture attacks on combinations, like the mix-and-match attacks above, our attacker faces multiple honest signers, instead of only a single signer as in all previous works.[3]

Our main modification is the notion of "freshness" for the adversary's forgery attempt. More precisely, we define a "minimal" closure set of all trivial message combinations for which the adversary can easily assemble a valid aggregate out of the data in the attack. For example, the closure contains any message set from an aggregation query but where the adversary trivially adds messages authenticated by corrupt parties. Every message set not appearing in this closure is then declared as fresh. Unlike previous definitions the forgery combination in the mix-and-match attack above is still fresh according to this definition.

History-Free Sequential Aggregation. It is not known how and if our general security models can be satisfied; even if we make the excessive assumption of some shared and synchronized information between the nodes, like a counter, we show that we can only achieve a slightly weaker version. Yet, the discussion still shows the limitations of current schemes and we can transfer the main ideas to the important case of *sequential* aggregation where, e.g., a sensor receives some data, performs some operation, and forwards the new data to the next node. With the corresponding adaptations of our security notion —and noting that the attacks above are in fact carried out in the sequential setting— it follows that our security guarantees also go beyond the current models for sequential schemes.

Yet, we even consider a stronger model than pure sequential aggregation. Recall that the proposal of Katz and Lindell supports the aggregation of the data independently of the order of the parties and the aggregating algorithm is key less. The

[3] It is tempting to assume that playing against a single honest user would suffice by a standard guessing strategy. However, the mix-and-match attack shows that we may not exploit a weakness in the tagging algorithm, but rather take advantage of the aggregation of tags by several honest parties or of the structure of the aggregate itself.

gist of known non-sequential schemes is that the aggregation algorithm computes the new data without inspection of previous messages. To preserve as much of this idea for sequential aggregate MACs we introduce the notion of *history-free* aggregation where the aggregation only depends on the aggregate-so-far and the local message. It is clear that the previous aggregate enters the computation and that this value itself carries (more or less explicit) information about previous messages. Due to the size restriction for aggregates, though, this information is limited. In this sense it is understood that history-free schemes only deny explicit access to previous messages. History-free sequential aggregation is a desirable goal from an efficiency point of view. It allows for example incremental compression of the message sequence without local decompression for each aggregation. This property is especially worthwhile for cases of MANETs where each computation effects on the battery life of the nodes.

In the history-free sequential case we provide solutions meeting our high standards. Our secure construction is based on any pseudorandom permutation (PRP) like AES. The idea here is to carefully chain the tags. In each aggregation step one basically computes a CBC-MAC of the current message concatenated with the previous tag (where we need the properties of the PRP only in the last mixing step). Hence, each aggregation step essentially requires the computation of a MAC plus one invocation of a PRP.

Related Work. Most works about secure aggregation follow the security model of Boneh et al. [2] and Lysyanskaya et al. [6]. The only exception is the recent work by Boldyreva et al. [3] which sketches a possible stronger security notion covering attacks on sequential schemes in which the adversary outputs a prefix of some aggregation query (and then possibly appends further iterations of corrupt players). But their model does not discuss more advanced attacks like "gluing" together aggregates, nor do they provide provably secure solutions for their model, whereas we show how to make schemes secure against very powerful attacks.

We note that the notion of sequential aggregate signed data, recently proposed by Neven [7], also aims at efficiency gains, but focuses on communication complexity instead of computational complexity. For such sequential aggregate signed data only the aggregate (being of roughly the same size as the messages) is passed to the next round. However, according to the definition this aggregate allows to recover the previously signed messages and Neven's solution indeed extracts all these messages for each aggregation step. In this sense, his solution is therefore still not history-free, unlike our construction for MACs.

Organization. In Section 2 we recall the notion of aggregate MACs. We introduce our model for aggregation unforgeability in Section 3. For our constructions we switch to the (history-free) sequential case in Section 4. There, we define history-free sequential aggregate MACs, discuss aggregation unforgeability in this case and finally we present our construction based on the general assumptions.

2 Non-Sequential Aggregation of MACs

Roughly speaking, an aggregate MAC is a single tag, called the aggregate, of q different users on q different messages such that the aggregate has nearly the same size as an ordinary tag. The well known definition of MACs and their security are given in Appendix B.

2.1 Definition

Definition 1 (Aggregate MACs). *An aggregate message authentication code* $\mathsf{Agg} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf}, \mathsf{Agg}, \mathsf{AVf})$ *is a tuple of efficient algorithms such that:*

Key Generation. *The algorithm* KGen *takes the security parameter* 1^n *and returns for a particular sender a pair* $(sk_{\mathsf{id}}, \mathsf{id})$ *where* sk_{id} *is a key and* id *is an identifier.*

Authentication, Mac Verification. Mac *and* Vf *are defined as in a standard message authentication scheme.*

Aggregation. *Upon input of two sets of message/identifier pairs* $M_1 = \{(m_1^1, \mathsf{id}_1^1), \ldots, (m_{\ell_1}^1, \mathsf{id}_{\ell_1}^1)\}$ *and* $M_2 = \{(m_1^2, \mathsf{id}_1^2), \ldots, (m_{\ell_2}^2, \mathsf{id}_{\ell_2}^2)\}$ *and associated tags* σ_1 *and* σ_2*, algorithm* Agg *outputs a new tag* σ*.*

Aggregate Verification. *Algorithm* AVf *accepts as input a set of key/identifier pairs* $\mathsf{sk} = \{(sk_1, \mathsf{id}_1), \ldots, (sk_t, \mathsf{id}_t)\}$*, a set of message/identifier pairs* $M = \{(m_1, \mathsf{id}_1'), \ldots, (m_\ell, \mathsf{id}_\ell')\}$ *and a tag* σ*. This algorithm returns a bit.*

An aggregate message authentication scheme is complete *if the following conditions hold:*

- *For any* $n \in \mathbb{N}$*, any* $(sk_{\mathsf{id}}, \mathsf{id}) \leftarrow \mathsf{KGen}(1^n)$*, any message* $m \in \mathcal{M}_n$*, we have* $\mathsf{Vf}(sk_{\mathsf{id}}, m, \mathsf{Mac}(sk_{\mathsf{id}}, m)) = 1$*.*
- *Let* M_1 *and* M_2 *be two sets of message/identifier pairs with* $M_1 \cap M_2 = \emptyset$*, let* sk_1 *as well as* sk_2 *be a set of keys, and let* $M = M_1 \cup M_2$ *and* $\mathsf{sk} = \mathsf{sk}_1 \cup \mathsf{sk}_2$*. If* $\mathsf{AVf}(\mathsf{sk}_1, M_1, \sigma_1) = 1$ *and* $\mathsf{AVf}(\mathsf{sk}_2, M_2, \sigma_2) = 1$ *then* $\mathsf{AVf}(\mathsf{sk}, M, \mathsf{Agg}(M_1, M_2, \sigma_1, \sigma_2)) = 1$*.*

2.2 Security Model and an Instantiation

The security model for aggregate MACs is closely related to the one for aggregate signatures [2]. The only technical difference results from the shared-key setting. Here, an adversary has access to two different oracles. The first oracle, the corruption oracle $\mathsf{Corrupt}(\mathsf{sk}, \cdot)$, returns on input id the corresponding secret key sk_{id}. The second oracle $\mathsf{OMac}(\mathsf{sk}, \cdot)$ allows the adversary to compute MACs for messages and keys of its choice. This oracle is initialized with a set of keys $\mathsf{sk} = ((sk_1, \mathsf{id}_1), \ldots, (sk_\ell, \mathsf{id}_\ell))$ and takes as input a message/identifier pair (m, id), it returns $\sigma \leftarrow \mathsf{Mac}(sk_{\mathsf{id}}, m)$. The adversary is successful if it outputs a set of message/identifier pairs $M = \{(m_1, \mathsf{id}_1), \ldots, (m_\ell, \mathsf{id}_\ell)\}$ and valid tag σ such that there exists at least one pair $(m_{i^*}, \mathsf{id}_{i^*}) \in M$ where id_{i^*} has not been corrupted, nor has $\mathcal{A}$ queried the MAC oracle on input $(m_{i^*}, \mathsf{id}_{i^*})$.

Definition 2 (Unforgeability). *An aggregate message authentication code scheme* $\mathsf{Agg} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf}, \mathsf{Agg}, \mathsf{AVf})$ *is* unforgeable *if for any efficient algorithm* $\mathcal{A}$ *the probability that the experiment* $\mathsf{AggForge}_{\mathcal{A}}^{\mathsf{Agg}}$ *evaluates to* 1 *is negligible (as a function of* n*), where*

Experiment $\mathsf{AggForge}_{\mathcal{A}}^{\mathsf{Agg}}(n)$
$(sk_1, \mathsf{id}_1), \ldots, (sk_t, \mathsf{id}_t) \leftarrow \mathsf{KGen}(1^n)$
$\mathsf{sk} \leftarrow ((sk_1, \mathsf{id}_1), \ldots, (sk_t, \mathsf{id}_t))$
$(M, \sigma) \leftarrow \mathcal{A}^{\mathsf{Corrupt}(\mathsf{sk}, \cdot), \mathsf{OMac}(\mathsf{sk}, \cdot)}(\mathsf{id}_1, \ldots, \mathsf{id}_t)$
Return 1 iff $\mathsf{AVf}(\mathsf{sk}, M, \sigma) = 1$ *and there exists a pair* $(m_{i^*}, \mathsf{id}_{i^*}) \in M$ *such that* $\mathcal{A}$ *never queried* $\mathsf{Corrupt}$ *about* id_{i^*} *and* $\mathcal{A}$ *never invoked* OMac *on input* $(m_{i^*}, \mathsf{id}_{i^*})$*.*

Instantiation According to Katz-Lindell. The authors also proposed the following provably secure construction, which we call XOR-AMAC. The aggregate message authentication code scheme simply computes XOR of all tags.

Construction 1. *Let* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ *be a deterministic message authentication code and define* $\mathsf{Agg} = (\mathsf{KGen}_{\mathsf{KL}}, \mathsf{Mac}_{\mathsf{KL}}, \mathsf{Vf}_{\mathsf{KL}}, \mathsf{Agg}_{\mathsf{KL}}, \mathsf{AVf}_{\mathsf{KL}})$ *through the following algorithms:*

Key Generation. *Algorithm* $\mathsf{KGen}_{\mathsf{KL}}(1^n)$ *executes for each user independently the key generation algorithm of the underlying MAC scheme* $sk \leftarrow \mathsf{KGen}(1^n)$ *and picks an identifier* $\mathsf{id} \leftarrow \{0,1\}^n$ *at random. It returns the pair* $(sk_{\mathsf{id}}, \mathsf{id})$.

Authentication, Verification. *Defined as in the underlying mac scheme.*

Aggregation. *Upon input two sets* M_1 *and* M_2 *of message/identifier pairs and two tags* σ_1 *and* σ_2 *the algorithm outputs* $\sigma = \sigma_1 \oplus \sigma_2$.

Aggregate Verification. $\mathsf{AVf}_{\mathsf{KL}}$ *takes as input a set of keys* $\mathsf{sk} = ((sk_1, \mathsf{id}_1), \ldots, (sk_\ell, \mathsf{id}_\ell))$, *a set* $M = \{(m_1, \mathsf{id}'_1), \ldots, (m_\ell, \mathsf{id}'_\ell)\}$ *of message/identifier pairs, and a tag* σ. *This algorithm* $\mathsf{AVf}_{\mathsf{KL}}$ *computes* $\sigma' = \bigoplus_{i=1}^{\ell} \mathsf{Mac}(sk_{\mathsf{id}_i}, m_i)$ *and outputs 1 if and only if* $\sigma' = \sigma$.

3 Aggregation Unforgeability for Non-Sequential MACs

In this section we first address the non-ordered case of aggregation. As discussed in the introduction, we introduce a security model that captures the broad class of mix-and-match attacks. It is clear that simple countermeasures like prepending the identifier of the user do not prevent theses attacks. Another approach might be to let the sender choose a nonce and have each intermediate user sign this nonce together with the message. The receiver only accepts aggregates for fresh nonces. This approach has some disadvantages, though. First, if the party choosing the nonces is controlled by the adversary, then a nonce may re-appear for several MAC generations.[4] Secondly, ad-hoc networks are highly dynamic. Thus, a node may receive an aggregate more than once (due to undesired loops in the route). Another disadvantage is that the receivers have to keep state. Similar arguments hold also for timing-based or counter-based solution. Nevertheless, we show in Appendix C a counter-based solution.

3.1 Security Model

We propose a stronger definition of unforgeability which we call aggregation-unforgeability. It follows the idea that the adversary is considered successful if he manages to find a valid aggregate for a message set which is not a straightforward combination of previous queries (or aggregates augmented by contributions of corrupt parties).

Regarding aggregate MACs, the main difference to the previous model is manifested in the fact that we grant the adversary in our model an additional *aggregation oracle* returning aggregates for sets of messages. The aggregation oracle, denoted by OAgg, is initialized with the key/identity pairs (sk_i, id_i) of all parties, takes as input a set of message/identifier pairs $\{(M_1, \mathsf{id}_1), \ldots, (M_k, \mathsf{id}_k)\}$ and returns an aggregate

[4] Note that letting each party choose a nonce and append it to the aggregate would lead the idea behind aggregation ad absurdum.

MAC σ for these data. We remark that the aggregation oracle only aggregates for honest parties, i.e., where the corresponding keys were not corrupted by the adversary; for corrupted parties the adversary must later add the values himself.

To express that the final output of the adversary is not a trivial combination of the results of the queries, we define a closure of the queries that contains all of these trivial combinations. For this definition we need the following notations. By Q_{Mac} we denote the set of queries of the adversary to the OMac oracle, by Q_{Agg} the set of queries to the aggregation oracle OAgg, and by Q_{Cor} the set of corruption queries. As a very basic example consider the classical unforgeability notion of MACs (one party only). Then the sets Q_{Agg} and Q_{Cor} are empty and Q_{Mac} contains exactly the queries to the MAC oracle. Here, trivial attacks are those where the adversary's forgery is for one of the previously queried messages from Q_{Mac}, i.e., the closure consists exactly of the queried messages.

In the case of aggregation the adversary can assemble more trivial message sets from its data. For example, if the adversary has obtained the aggregated MAC for a pair of messages and identities $\mathsf{id}_1, \mathsf{id}_2$, and knows the MAC for a third honest party id_3, then it can run the public aggregation algorithm to derive a valid MAC for the three messages. Analogously, the adversary can add corrupt parties easily by computing individual MACs for these parties and then aggregating them to a previous result. Our definition follows this idea, basically saying that the closure of all trivial combinations contains aggregation queries to which we add individual MAC queries and corrupt parties as well as further aggregation queries.

Consider as an instructive example a sensor network monitoring temperature differences, where deviations of 2°F between adjacent sensors would trigger an alarm. Suppose for simplicity that the network only consists of two nodes, one (called 'master') being closer to the base station and forwarding the data from the other node (called 'slave') to the station. When using an aggregation scheme the master sensor receives an aggregate for a temperature from the slave, "inserts" its authentication data for its temperature and forwards the temperatures and the new aggregate to the base.

If the adversary sees the aggregated MACs to the innocuous measurements $(70°\mathrm{F}, 70°\mathrm{F})$, $(69°\mathrm{F}, 70°\mathrm{F})$, and $(70°\mathrm{F}, 71°\mathrm{F})$, then

$$Q_{\mathsf{Agg}} = \big\{\{(70, \mathsf{id}_1), (70, \mathsf{id}_2)\}, \{(69, \mathsf{id}_1), (70, \mathsf{id}_2)\}, \{(70, \mathsf{id}_1), (71, \mathsf{id}_2)\}\big\}$$

for identities id_1 = 'slave' and id_2 = 'master'. Assume that there is a third party id_3 which is honest and for which the adversary has obtained an individual MAC $Q_{\mathsf{Mac}} = \{(65, \mathsf{id}_3)\}$ and that there is no corrupt party, $Q_{\mathsf{Cor}} = \emptyset$. Then the closure would be

$$\begin{aligned}
&\mathsf{Closure}(Q_{\mathsf{Mac}}, Q_{\mathsf{Agg}}, Q_{\mathsf{Cor}}) \\
&= \big\{\{(65, \mathsf{id}_3)\}, \\
&\quad \{(70, \mathsf{id}_1), (70, \mathsf{id}_2)\},\ \{(69, \mathsf{id}_1), (70, \mathsf{id}_2)\},\ \{(70, \mathsf{id}_1), (71, \mathsf{id}_2)\}, \\
&\quad \{(69, \mathsf{id}_1), (70, \mathsf{id}_1), (70, \mathsf{id}_2)\},\ \{(70, \mathsf{id}_1), (70, \mathsf{id}_2), (71, \mathsf{id}_2)\}, \\
&\quad \{(70, \mathsf{id}_1), (70, \mathsf{id}_2), (65, \mathsf{id}_3)\},\ \{(69, \mathsf{id}_1), (70, \mathsf{id}_2), (65, \mathsf{id}_3)\},\ \{(70, \mathsf{id}_1), (71, \mathsf{id}_2), (65, \mathsf{id}_3)\}, \\
&\quad \{(69, \mathsf{id}_1), (70, \mathsf{id}_1), (70, \mathsf{id}_2), (71, \mathsf{id}_2)\}, \dots \big\}
\end{aligned}$$

Note that we do not treat sets where an identity appears multiple times in any special way. However, such forgery attempts can be easily thwarted by having the verifier

check that all identities are distinct. We remark that the pair $\{(69, \mathsf{id}_1), (71, \mathsf{id}_2)\}$ is not a member of the closure (containing only the three originally queries as entries with two elements), but for which the adversary can for example in the Katz-Lindell scheme easily obtain a valid aggregate by adding the aggregates for the three measurements. The aggregate for this pair, even though not forging an individual MAC, would nonetheless trigger an alarm because of the temperature distance.

Definition 3 (Closure of $\mathcal{A}$'s queries). *The closure* $\mathsf{Closure}$ *of $\mathcal{A}$'s queries Q_{Mac}, Q_{Agg} and Q_{Cor} is defined as*

$$\mathsf{Closure}(Q_{\mathsf{Mac}}, Q_{\mathsf{Agg}}, Q_{\mathsf{Cor}}) = \left\{ \bigcup_{M_A \in A} M_A \cup M_M \cup M_C \;\middle|\; A \subseteq Q_{\mathsf{Agg}},\, M_M \subseteq Q_{\mathsf{Mac}},\, M_C \subseteq \bigcup_{\mathsf{id} \in Q_{\mathsf{Cor}}} \{(m, \mathsf{id}) \,|\, m \in \mathcal{M}_n\} \right\}$$

with $\mathcal{M}_n$ denoting the message space for the security parameter n.

With our definition of the closure we get the following definition for aggregation-unforgeable MAC schemes.

Definition 4. *An aggregate message authentication code scheme* $\mathsf{Agg} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf}, \mathsf{Agg}, \mathsf{AVf})$ *is* aggregation-unforgeable *if for any efficient algorithm $\mathcal{A}$ the probability that the experiment* $\mathsf{AggForge}_{\mathcal{A}}^{\mathsf{Agg}}$ *evaluates to* 1 *is negligible (as a function of n), where*

Experiment $\mathsf{AggForge}_{\mathcal{A}}^{\mathsf{Agg}}(n)$
$(sk_1, \mathsf{id}_1), \ldots, (sk_t, \mathsf{id}_t) \leftarrow \mathsf{KGen}(1^n)$
$\mathsf{sk} \leftarrow ((sk_1, \mathsf{id}_1), \ldots, (sk_t, \mathsf{id}_t))$
$(M, \sigma) \leftarrow \mathcal{A}^{\mathsf{Corrupt}(\mathsf{sk},\cdot),\mathsf{OMac}(\mathsf{sk},\cdot),\mathsf{OAgg}(\mathsf{sk},\cdot)}(\mathsf{id}_1, \ldots, \mathsf{id}_t)$
Return 1 iff $\mathsf{AVf}(\mathsf{sk}, M, \sigma) = 1$ *and* $M \notin \mathsf{Closure}(Q_{\mathsf{Mac}}, Q_{\mathsf{Agg}}, Q_{\mathsf{Cor}})$.

3.2 Relationship to the Model of Katz-Lindell

We first prove formally the fact that aggregation-unforgeability implies unforgeability. Then we separate the notion by showing that the aggregate MAC scheme shown in Construction 1 is aggregation-forgeable.

Proposition 1. *Every aggregation-unforgeable message authentication code is also unforgeable.*

Proof. Let $\mathsf{Agg} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf}, \mathsf{Agg}, \mathsf{AVf})$ be an aggregation-unforgeable message authentication scheme. Suppose towards contradiction that there exists an adversary $\mathcal{A}$ breaking security of Agg. Then we show how to build an algorithm $\mathcal{B}$ against aggregation-unforgeability. This algorithm executes a black-box simulation of $\mathcal{A}$ and answers each oracle query with its own oracles. Finally, $\mathcal{A}$ stops, outputting a pair (M, σ) which $\mathcal{B}$ returns as its forgery.

Algorithm $\mathcal{B}$ performs a perfect simulation from $\mathcal{A}$'s point of view, and since $\mathcal{A}$ is efficient $\mathcal{B}$ is also efficient. To see that the forgery is valid, note that Q_{Agg} is empty because $\mathcal{A}$ performs the aggregation queries locally. Recall that $\mathcal{A}$ only succeeds if there exists at least one pair $(m_{\mathsf{id}_{i^*}}, \mathsf{id}_{i^*}) \in M$ such that $\mathcal{A}$ never queried $\mathsf{Corrupt}$ about id_{i^*} and never invoked OMac on $(m_{\mathsf{id}_{i^*}}, \mathsf{id}_{i^*})$. Thus, the forgery is not in the closure and $\mathcal{B}$ succeeds whenever $\mathcal{A}$ returns a valid forgery. □

In the following we separate the notions showing that Construction 1 is "aggregation-forgeable". The basic idea follows the example that we discussed in the previous section and is that $\mathcal{A}$ successfully recombines real subsets of queries to the AMac oracle. Thus, $\mathcal{A}$'s answer M is a set which contains has never been sent to the oracle OAgg.

Proposition 2. *If there exists a deterministic message authentication code where the message-space $\mathcal{M}_n$ contains at least four distinct messages, then the aggregate message authentication code defined in Construction 1 is* not *aggregation-unforgeable.*

Proof. The adversary $\mathcal{A}$ forging the aggregate MAC (cf. Definition 4) gets as input $(\mathsf{id}_1, \ldots, \mathsf{id}_t)$ and works as follows: It first picks two identifiers at random from the list, $(\mathsf{id}_1, \mathsf{id}_2)$, chooses randomly four messages $m_1, m_2, m_3, m_4 \leftarrow \mathcal{M}_n$ and sets $M_1 \leftarrow ((m_1, \mathsf{id}_1)(m_2, \mathsf{id}_2))$, $M_2 \leftarrow ((m_1, \mathsf{id}_1)(m_3, \mathsf{id}_2))$ and, $M_3 \leftarrow ((m_4, \mathsf{id}_1)(m_2, \mathsf{id}_2))$. This algorithm then invokes the aggregation oracle three times:

$$\sigma_1 \leftarrow \mathsf{OAgg}(\mathsf{sk}, M_1) \quad \text{and} \quad \sigma_2 \leftarrow \mathsf{OAgg}(\mathsf{sk}, M_2) \quad \text{and} \quad \sigma_3 \leftarrow \mathsf{OAgg}(\mathsf{sk}, M_3).$$

It returns $(M, \sigma) \leftarrow (((m_4, \mathsf{id}_1), (m_3, \mathsf{id}_2)), (\sigma_1 \oplus \sigma_2 \oplus \sigma_3))$.

For the analysis it is easy to see that $\mathcal{A}$ is efficient. The forgery is valid since

$$\begin{aligned}
\sigma = \sigma_1 \oplus \sigma_2 \oplus \sigma_3 &= \mathsf{OAgg}(\mathsf{sk}, M_1) \oplus \mathsf{OAgg}(\mathsf{sk}, M_2) \oplus \mathsf{OAgg}(\mathsf{sk}, M_3) \\
&= \mathsf{Mac}(sk_{\mathsf{id}_1}, m_1) \oplus \mathsf{Mac}(sk_{\mathsf{id}_2}, m_2) \oplus \mathsf{Mac}(sk_{\mathsf{id}_1}, m_1) \oplus \mathsf{Mac}(sk_{\mathsf{id}_2}, m_3) \\
&\quad \oplus \mathsf{Mac}(sk_{\mathsf{id}_1}, m_4) \oplus \mathsf{Mac}(sk_{\mathsf{id}_2}, m_2) \\
&= \mathsf{Mac}(sk_{\mathsf{id}_1}, m_4) \oplus \mathsf{Mac}(sk_{\mathsf{id}_2}, m_3)
\end{aligned}$$

holds. Furthermore $\mathcal{A}$ neither queried the corruption oracle, nor invoked $\mathsf{OAgg}(\mathsf{sk}, \cdot)$ on the tuple $((m_4, \mathsf{id}_1), (m_3, \mathsf{id}_2))$. □

4 History-Free Sequential Aggregate MACs

In this section we introduce the notion of history-free sequential aggregation and adapt the desired security model of aggregation-unforgeability to the new scenario. We then present our sequential aggregate MAC scheme based on an underlying deterministic MAC.

4.1 Definition of Sequential Aggregate MACs

In an aggregate MAC scheme the tags are computed independently by each sender and are then combined into a single aggregate tag. Therefore, the aggregation can be performed even by an unrelated party since the process does not require knowledge of the secret keys. In contrast, in a sequential aggregate MAC schemes the authentication and aggregation is a combined operation. Each sender gets as additional input an aggregate-so-far σ' and transforms that tag into a new aggregate σ which includes the authentication of a message of his choice. We write $M||(m, \mathsf{id})$ for the resulting sequence of message-identifier pairs (where the pair (m, id) is appended to the previous pairs).

Definition 5. *A* sequential aggregate message authentication code *scheme is a tuple of efficient algorithms* $\mathsf{SAGG} = (\mathsf{SKGen}, \mathsf{Mac}, \mathsf{Vf}, \mathsf{SMac}, \mathsf{SVf})$ *such that*

Key generation. SKGen *takes as input the security parameter* 1^n *and returns a key* sk_{id} *together with an identity* id.

Authentication, Verification. *Defined as in a standard MAC scheme.*

Aggregate Tagging. *Algorithm* SMac *accepts as input a key* sk_{id}*, a message* $m \in \mathcal{M}_n$*, an aggregate-so-far* tag σ' *and a sequence of message/id pairs* $M = ((m_1, \mathsf{id}_1), \ldots, (m_t, \mathsf{id}_t))$*. It outputs a new aggregate MAC* σ.

Verification algorithm. SVf *takes as input a set of keys* $\mathsf{sk} = \{sk_{\mathsf{id}_1}, \ldots, sk_{\mathsf{id}_\ell}\}$*, a tuple of messages/identifier pairs* $M = ((m_1, \mathsf{id}_1), \ldots, (m_t, \mathsf{id}_t))$ *as well as an alleged sequential aggregate tag* σ *and outputs a bit.*

A sequential aggregate MAC scheme is complete *if*

- *(Single-MAC Correctness) For any pair* $(sk_{\mathsf{id}}, \mathsf{id}) \leftarrow \mathsf{SKGen}(1^n)$*, any message* $m \in \mathcal{M}_n$ *and any* $\sigma \leftarrow \mathsf{Mac}(sk_{\mathsf{id}}, m)$*, it holds that* $\mathsf{Vf}(sk_{\mathsf{id}}, m, \sigma) = 1$.
- *(Aggregation Correctness) For all pairs* $(sk_{id}, \mathsf{id}) \leftarrow \mathsf{SKGen}(1^n)$*, all messages* $m \in \mathcal{M}_n$*, for any set of message/identifier pairs* $M = \{(m_1, \mathsf{id}_1), \ldots, (m_\ell, \mathsf{id}_\ell)\}$ *(where* $(m_i, \mathsf{id}_i) \in \mathcal{M}_n \times \{0,1\}^n$ *for all* $i = 1, \ldots, \ell$*), any set of keys* sk *and any tag* $\sigma' \in \mathcal{R}_n$ *with* $\mathsf{SVf}(\mathsf{sk}, M, \sigma') = 1$ *we require that for all* $\sigma \leftarrow \mathsf{SMac}(sk_{\mathsf{id}}, m, \sigma', M)$ *it holds that* $\mathsf{SVf}((\mathsf{sk}||sk_{\mathsf{id}}), M||(m, id_{\mathsf{id}}), \sigma) = 1$.

A common approach to build sequential aggregate *signature* schemes is to verify the validity of an received aggregate-so-far before computing the new aggregate. Often, the aggregation algorithm even includes the previous messages in its computations. In the private key setting, however, verification of the aggregate may not be possible as nodes do not share all keys. Moreover, compared with non-sequential schemes, where the aggregation process does not depend on the previous messages, this is a main drawback of sequential schemes (especially from an efficiency point of view). The idea of history-free sequential aggregation is to overcome that restriction by requiring that the aggregation only depends on the aggregate-so-far and the local message.

Definition 6 (History Freeness). *A sequential aggregate message authentication scheme* $\mathsf{SAGG} = (\mathsf{SKGen}, \mathsf{Mac}, \mathsf{Vf}, \mathsf{SMac}, \mathsf{SVf})$ *is called* history-free *if there exists an efficient algorithm* SMac_{hf} *such that* $\mathsf{SMac}_{hf}(\cdot,\cdot,\cdot) = \mathsf{SMac}(\cdot,\cdot,\cdot,M)$ *for all* M.

In the sequel we often identify $\mathsf{SMac}_{\mathsf{hf}}$ with SMac and simply omit M from the input of SMac.

4.2 Security Model

A sequential aggregate MAC is called *aggregation-unforgeable*, if any efficient adversary $\mathcal{A}$ succeeds in the following two-phase experiment only with negligible probability. In the first phase, the adversary has access to a corrupt oracle $\mathsf{Corrupt}$, and can obtain the secret keys of senders of his choice. As soon as $\mathcal{A}$ queries its sequential aggregate MAC oracle SeqAgg, the corruption phase has ended and the adversary $\mathcal{A}$ is not allowed to query the corrupt oracle again. The sequential aggregate MAC oracle SeqAgg takes as input a set $\mathsf{sk} = (sk_{\mathsf{id}_1}, \ldots, sk_{\mathsf{id}_\ell})$, an aggregate-so-far tag σ',

an ordered set $M = \{(m_1, \mathsf{id}_1), \ldots, (m_q, \mathsf{id}_q)\}$ of message identifier pairs and returns a (sequentially ordered) tag σ.

Before proposing the formal security model, we define the closure of all trivial combinations. We denote by Q_{Seq} the set of all query/answer tuples $((M, \sigma'), \sigma)$ that occur in $\mathcal{A}$'s interaction with the SeqAgg oracle and by Q_{Cor} we denote the set of all identities' that were queried to the $\mathsf{Corrupt}$ oracle.

We stress that in the context of sequential aggregate MACs given the adversary access to a MAC oracle is redundant. Each query to a (single) MAC oracle can easily simulated by calling the sequential aggregate oracle with the empty tag $\sigma_\varnothing$. Thus, the definition of the closure does not need the set Q_{Mac} of queries and responses from the MAC oracle (since this set in contained in Q_{Seq}).

Definition 7 (Sequential Closure of $\mathcal{A}$'s queries). *Let M be a set of message/identifier pairs, let Q_{Cor} and Q_{Seq} be the sets corresponding to the different oracle responses and let $m_\varnothing$ ($\sigma_\varnothing$) be the empty message (empty tag, respectively). Let $\mathsf{Trivial}_{Q_{\mathsf{Seq}}, Q_{\mathsf{Cor}}}$ be a recursive function of trivial combinations defined as*

$$\mathsf{Trivial}_{Q_{\mathsf{Seq}}, Q_{\mathsf{Cor}}}(M, \sigma) := \{M\} \cup \bigcup_{((\sigma, M'), \sigma') \in Q_{\mathsf{Agg}}} \mathsf{Trivial}_{Q_{\mathsf{Seq}}, Q_{\mathsf{Cor}}}(M || M', \sigma') \cup \bigcup_{\substack{\forall \bar{m}, \bar{\sigma} \\ \wedge id_i \in Q_{\mathsf{Cor}}}} \mathsf{Trivial}_{Q_{\mathsf{Seq}}, Q_{\mathsf{Cor}}}(M || (\bar{m}, id_i), \bar{\sigma}) \,.$$

We can now define the closure $\mathsf{Closure}$ *of $\mathcal{A}$'s queries Q_{Agg} and Q_{Cor} by recursively generating the trivial combinations starting from the empty message $m_\varnothing$ and empty tag $\sigma_\varnothing$ as described above:*

$$\mathsf{Closure}(Q_{\mathsf{Agg}}, Q_{\mathsf{Cor}}) := \{\mathsf{Trivial}_{Q_{\mathsf{Seq}}, Q_{\mathsf{Cor}}}(m_\varnothing, \sigma_\varnothing)\}.$$

With the definition of the sequential closure, we propose the following security model for sequential aggregate MACs.

Definition 8. *A sequential aggregate message authentication code scheme* $\mathsf{SAGG} = (\mathsf{SKGen}, \mathsf{Mac}, \mathsf{Vf}, \mathsf{SMac}, \mathsf{SVf})$ *is* aggregation-unforgeable *if for any efficient algorithm $\mathcal{A}$ (working in mode* COR, FOR*) the probability that the experiment* $\mathsf{SeqForge}_{\mathcal{A}}^{\mathsf{Agg}}$ *evaluates to* 1 *is negligible (as a function of n), where*

Experiment $\mathsf{SeqForge}_{\mathcal{A}}^{\mathsf{Agg}}(n)$
$(sk_1, \mathsf{id}_1), \ldots, (sk_t, \mathsf{id}_t) \leftarrow \mathsf{SKGen}(1^n)$
$\mathsf{sk} \leftarrow ((sk_1, \mathsf{id}_1), \ldots, (sk_t, \mathsf{id}_t))$
$\mathsf{st} \leftarrow \mathcal{A}^{\mathsf{Corrupt}(\mathsf{sk}, \cdot)}(\mathrm{COR}, \mathsf{id}_1, \ldots . \mathsf{id}_t)$ *// it is understood that $\mathcal{A}$ keeps state* st
$(M, \sigma) \leftarrow \mathcal{A}^{\mathsf{SeqAgg}(\mathsf{sk}, \cdot, \cdot)}(\mathrm{FOR}, \mathsf{st}, \mathsf{id}_1, \ldots, \mathsf{id}_t)$
Return 1 iff $\mathsf{id}_i \neq \mathsf{id}_j$ *for all* $i \neq j$ *and* $\mathsf{SVf}(\mathsf{sk}, M, \sigma) = 1$ *and*
$M \notin \mathsf{Closure}(Q_{\mathsf{Agg}}, Q_{\mathsf{Cor}})$.

Note that in the definition above the adversary $\mathcal{A}$ running in mode FOR has only access to the sequential aggregate MAC oracle an *not* to a tagging oracle Mac. We argue that this is redundant since the attacker is allowed to invoke SeqAgg on tags of its choice. Thus, $\mathcal{A}$ can query SeqAgg on arbitrary messages m together with the empty tag $\sigma_\varnothing$ and yields an ordinary tag for m.

5 Construction of History-Free Sequential MACs

The idea behind our construction is as follows. We again use a "chaining" approach in which we let the next aggregating party (with identity id) compute the next tag over its own message $M \in \{0,1\}^*$ and over the previous tag σ'. That is, $\tau \leftarrow \mathsf{Mac}(sk, M\|\sigma')$ for the deterministic algorithm $\mathsf{Mac}(sk,\cdot) : \{0,1\}^* \to \{0,1\}^n$. To preserve the order of the aggregating parties we let each party prepend its own identity id to the resulting tag $\sigma \leftarrow \mathsf{id}\|\tau$. Thus, the next party essentially computes a MAC for its own message, the identity of the previous party and the previous tagging result. Formally, prepending id enlarges the tag, yet in most applications the identity of the sending party is known anyway and does not need to be included explicitly.

Proving the security of the above approach leads to some difficulties. Namely, the adversary could potentially gain information from the final tag about an intermediate value, and could thus easily "shorten" such aggregation chains. To prevent this we assume that MAC itself is pseudorandom, ensuring that no such information is leaked.

We also need a special property of the MAC allowing us to "go backwards" in a chain: assume that an adversary successfully outputs a forged sequence by predicting one of the intermediate MACs correctly. Then, in order to break the security of the underlying MAC, we need to be able to undo the MAC computations afterwards and to access the intermediate MAC values. We add this *partial inversion* property as an requirement to the (pseudorandom) MAC and show that standard constructions like CMAC have this property and that one can easily build such MACs from pseudorandom permutations.

5.1 Properties of the MAC

Recall that we need two properties of the underlying MAC in order to make our construction work: pseudorandomness and partial inversion:

Definition 9 (Pseudorandom MAC). *A det. message authentication code* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ *is* pseudorandom *(or a* pseudorandom function*) if for any efficient algorithm* $\mathcal{D}$ *the value*

$$\left|\mathrm{Prob}\left[\mathcal{D}^{\mathsf{Mac}(sk,\cdot)}(1^n) = 1\right] - \mathrm{Prob}\left[\mathcal{D}^{f(\cdot)}(1^n) = 1\right]\right|$$

is negligible, where the probability in the first case is over $\mathcal{D}$*'s coin tosses and the choice of* $sk \leftarrow \mathsf{KGen}(1^n)$*, and in the second case over* $\mathcal{D}$*'s coin tosses and the choice of the random function* $f : \{0,1\}^* \to \{0,1\}^n$*.*

A pseudorandom function is called a pseudorandom permutation if it is also a permutation. Note that pseudorandom MACs are unforgeable, too.

Definition 10 (Partial Inversion). *A deterministic message authentication code* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ *is* partially invertible *if there exists an efficient algorithm* $\mathsf{PartInv}$ *which, for any security parameter* $n \in \mathbb{N}$*, any key* $sk \leftarrow \mathsf{KGen}(1^n)$*, any* $M = M'||m$ *for some* $m \in \{0,1\}^n$*, and any* $\sigma \in \{0,1\}^n$*, on input* (sk, M', σ) *returns a string* $\mathsf{m} \in \{0,1\}^n$ *such that* $\mathsf{Mac}(sk, M'\|\mathsf{m}) = \sigma$*.*

In the following we present two efficient constructions satisfying the definition of partial inversion. The first construction is CMAC (a security proof is given by Iwata

and Kurosawa under the name OMAC in [4]) which can be used for messages of fixed block length. The reason for not using CMAC for arbitrary input-lengths is that the desired block may not be aligned to the final n bits. The second construction uses a pseudorandom permutation and is applicable for messages of variable length.

A Solution Based on CMAC. If the length of the message/identifier pair is a positive multiple of the block size, then CMAC can be used as the underlying (pseudorandom) message authentication code (when a pseudorandom permutation PRP lies underneath). We first review CMAC briefly and show then that CMAC supports partial inversion.

The key generation algorithm of CMAC generates a pair of keys sk, sk_1 where sk_1 is derived from sk.[5] In order to compute a tag, the tagging algorithm takes as input a message $M = m_1 || \ldots || m_k \in \{0,1\}^{k \cdot n}$ and two keys sk, sk_1. It computes

$$c_i \leftarrow \mathsf{PRP}(sk, m_i \oplus c_{i-1}) \text{ for } i = 1, \ldots, k-1 \text{ (where } c_0 = 0^n),$$

and outputs the final tag as $\sigma \leftarrow \mathsf{PRP}(sk, c_{k-1} \oplus sk_1 \oplus m_k)$. Unforgeability and pseudorandomness follow from the security of CMAC for aligned inputs.

Lemma 1. *CMAC is partially invertible.*

Proof. In the following we describe the algorithm $\mathsf{PartInv}$ which takes as input a message $M' = m_1, \ldots, m_{k-1}$ (consisting of $k-1$ blocks $m_1, \ldots, m_{k-1}$ of n bits each) a pair of keys (sk, sk_1), and a tag σ. In the first step, this algorithm emulates the iteration of CMAC but omitting the last step, $c_i \leftarrow \mathsf{PRP}(sk, m_i \oplus c_{i-1})$ for $i = 1, \ldots, k-1$. Algorithm $\mathsf{PartInv}$ then decrypts the received tag $\tau \leftarrow \mathsf{PRP}^{-1}(sk, \sigma)$ and returns $\mathsf{m} \leftarrow c_{k-1} \oplus sk_1 \oplus \tau$. It is clear that this recovers an appropriate value m. □

A General Solution. In the following let $M = M' \| m \in \mathcal{M}_n$ be a message whose block length is *not* a positive multiple of the block size. We then present a suitable MAC scheme based on general assumptions. The main idea of the construction is to execute (the underlying deterministic) tagging algorithm $\tau \leftarrow \mathsf{Mac}(sk, M')$ on the first part M' of the message M and to compute a pseudorandom permutation on the value $\tau \oplus m$.

Construction 2. *Let* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ *be a deterministic message authentication code and* $\mathsf{PRP}(\cdot,\cdot)$ *be a pseudorandom permutation (where* Mac *for security parameter* n *produces* n*-bit outputs and* PRP *is also over* n*-bits for security parameter* n*). We define the procedures* $\mathsf{CKg}, \mathsf{CTag}$ *and* CVf *as follows:*

KeyGen. *The key generation algorithm* $\mathsf{CKg}(1^n)$ *generates a key* $sk \leftarrow \mathsf{KGen}(1^n)$*, chooses a key* $\mathsf{k_{PRP}} \in \{0,1\}^n$ *at random and returns* $(sk, \mathsf{k_{PRP}})$.

Tagging. $\mathsf{CTag}((sk, \mathsf{k_{PRP}}), M)$ *takes a message* $M = M' \| m$ *with* $M' \in \{0,1\}^*$ *and* $m \in \{0,1\}^n$ *as well as a key pair* $sk, \mathsf{k_{PRP}}$*. It computes* $\tau \leftarrow \mathsf{Mac}(sk, M')$ *and returns the value* $\mathsf{PRP}(\mathsf{k_{PRP}}, \tau \oplus m)$.

Verification. *The algorithm* $\mathsf{CVf}((sk, \mathsf{k_{PRP}}), M, \sigma)$ *returns to* 1 *iff* $\mathsf{CTag}((sk, \mathsf{k_{PRP}}), M) = \sigma$*, otherwise* 0.

[5] Note that CMAC deduces two keys sk_1 and sk_2 from sk. As in this construction the second key sk_2 is not required, we omit it here.

Note that we do not claim to be able to recover the full message $M\|\sigma'$ from a MAC $\tau \leftarrow \mathsf{Mac}(sk, M\|\sigma')$, but it suffices that we recover σ' *given* sk, M *and* σ. The following theorem proves formally the security and the partial inversion property of the construction.

Theorem 3. *If* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ *is an unforgeable message authentication code and* PRP *is a pseudorandom permutation, then Construction 2 is a pseudorandom, partially invertible message authentication code.*

We proof this theorem through the following two proposition, first showing that the resulting MAC scheme is secure and second its partial inversion.

Proposition 3. *If* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ *is an unforgeable message authentication code and* PRP *is a pseudorandom permutation, then Construction 2 is pseudorandom.*

The proof idea is to apply the well-known result that the composition of a (computational) almost universal function and a pseudorandom function remains pseudorandom (see, for example, [1]). This clearly yields a secure MAC. Hence, for our construction it suffices to show that the "inner" part of our MAC algorithm is computational almost universal. Before stating this result, we give a formal definition of computational almost universal MACs.

Definition 11. *A message authentication code* MAC *is called* computational almost universal (cAU) *if for any efficient algorithm* $\mathcal{A}$ *the probability that experiment* cAU *evaluates to 1 is negligible (as a function of* n*), where*

Experiment $\mathsf{cAU}_{\mathcal{A}}^{\mathsf{MAC}}(n)$
$sk \leftarrow \mathsf{KGen}(1^n)$
$(M_1, M_2) \leftarrow \mathcal{A}(1^n)$
Return 1 iff $M_1 \neq M_2$ *and* $\mathsf{Mac}(sk, M_1) = \mathsf{Mac}(sk, M_2)$.

Lemma 2. *For an unforgeable deterministic message authentication codes* $\mathsf{MAC}' = (\mathsf{KGen}', \mathsf{Mac}', \mathsf{Vf}')$ *the algorithm* $\mathsf{Mac}'(sk, M') \oplus m$ *for* $M = M'||m$ *is computational almost universal (for key generation* $sk \leftarrow \mathsf{KGen}'(1^n)$*).*

Proof. To prove this lemma first consider the case that we have $M_1' = M_2'$ for a successful adversarial output $M_1 = M_1'||m_1$, $M_2 = M_2'||m_2$. Then it must hold that $m_1 \neq m_2$, implying that $\mathsf{Mac}'(sk, M_1') \oplus m_1 \neq \mathsf{Mac}'(sk, M_2') \oplus m_2$ for the deterministic algorithm Mac'. Hence assume from now on that there exists an algorithm $\mathcal{A}$ breaking the almost universal property of the MAC scheme proposed in Construction 2 with noticeable probability for $M_1' \neq M_2'$. We then build an algorithm $\mathcal{B}$, against the underlying MAC scheme MAC', which executes $\mathcal{A}$ in a black-box way and works as follows. $\mathcal{B}$ gets as input the security parameter 1^n, has access to an tagging oracle $\mathsf{Mac}(sk, \cdot)$ and initiates $\mathcal{A}$ on input 1^n. At the end of the simulation $\mathcal{A}$ outputs two messages $M_1 = M_1'||m_1$, $M_2 = M_2'||m_2$. $\mathcal{B}$ invokes its MAC oracle $\mathsf{Mac}(sk, \cdot)$ on M_1' and gets σ_1' and outputs $(M_2', \sigma_1' \oplus m_1 \oplus m_2)$.

For the analysis note that $\mathcal{B}$ is efficient since $\mathcal{A}$ is efficient. Furthermore, given that $\mathcal{A}$ produces a collision M_1, M_2 with $M_1' \neq M_2'$, adversary $\mathcal{B}$ succeeds in producing a forgery for a new message since it queries its oracle only once about $M_1' \neq M_2'$ and the derived tag is obviously valid. □

Concerning partial inversion, we have:

Proposition 4. *The message authentication code defined in Construction 2 is partially invertible.*

Proof. The construction supports partial inversion: The algorithm $\mathsf{PartInv}$ takes as input a pair of keys $(sk, \mathsf{k_{PRP}})$, a string M' and a tag σ. It computes $\tau \leftarrow \mathsf{Mac}(sk, M')$, $c \leftarrow \mathsf{PRP}^{-1}(\mathsf{k_{PRP}}, \sigma)$ and returns $\mathsf{m} \leftarrow \tau \oplus c$. It is now easy to see that this output is valid. □

5.2 Construction

Construction 4. *Let* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ *be a deterministic MAC. Let* $\mathsf{SAM} = (\mathsf{SeqKg}, \mathsf{SeqAgg}, \mathsf{SeqAggVf})$ *be as follows:*

Key Generation. *The key generation algorithm* SeqKg *takes as input the security parameter* 1^n*, picks an identifier at random* $\mathsf{id} \in \{0,1\}^n$*, executes the key generation algorithm of the underlying MAC scheme* $sk \leftarrow \mathsf{KGen}(1^n)$ *and returns the pair* (sk, id)*.*

Sequential Aggregation. *The algorithm* $\mathsf{SeqAgg}(sk, M, \mathsf{id}, \sigma')$ *takes as input a private key* sk*, a message* $M \in \{0,1\}^*$*, and a (sequentially aggregated) tag* σ'*. It executes the underlying tagging algorithm* $\tau \leftarrow \mathsf{Mac}(sk, M \| \sigma')$ *and outputs* $\sigma \leftarrow \mathsf{id} \| \tau$*. (For* $\sigma_0 = \emptyset$ *simply run the MAC algorithm on* M *only.)*

Aggregate Verification. *The input of algorithm* $\mathsf{SeqAggVf}$ *is a sequence of keys* $\mathsf{sk} = (sk_1, \ldots, sk_\ell)$*, a tag* σ *and sequences* $\mathbf{M} = (M_1, \ldots, M_\ell)$ *and* $\mathbf{id} = (\mathsf{id}_1, \ldots, \mathsf{id}_\ell)$ *of messages and identifiers. If any key in* sk *appears twice then return* 0*. Otherwise compute for* $i = 1, \ldots, \ell$*,* $\sigma_i \leftarrow \mathsf{id}_i \| \mathsf{Mac}(sk_i, M_i || \sigma_{i-1})$ *, with* $\sigma_0 \leftarrow \emptyset$*. Return* 1 *iff* $\sigma_\ell = \sigma$*, otherwise* 0*.*

The following theorem captures the security of our construction:

Theorem 5. *If* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ *is a pseudorandom, partially invertible message authentication code then Construction 4 is a history-free, aggregation-unforgeable sequential aggregate message authentication code scheme.*

More precisely, we show that, letting t be the number of parties and Q denote the number of aggregation queries, each of L message-identity pairs at most, the probability that an adversary breaks the aggregate MAC scheme is bounded from above by $3t(Q+1)^2L^2$ times the probability of breaking the underlying MAC, plus the advantage of breaking the pseudorandomness of the MAC (in both cases with comparable running time).

The idea of our proof is as follows. When the adversary eventually outputs a forgery attempt there must be a subsequence which is not assembled out of seen values or values of corrupt parties. In particular, this *target sequence* contains only values of honest parties (see Figure 1). We first show that there are no collisions among aggregates output by honest parties (else one could use this collision to forge MACs). It follows that this target sequence cannot be a suffix of an aggregation query because the identity in the forgery attempt must be different from the identity in the corresponding aggregation query (else we would have found a collision when entering the target sequence). The target sequence cannot be a prefix of a previous aggregation query because the pseudorandomness of the MAC hides all information

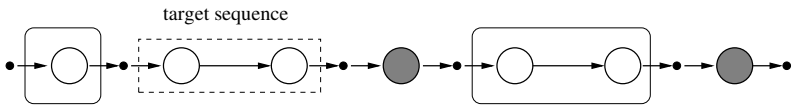

Fig. 1: Example of a target sequence. Shaded circles denote corrupt parties and boxes correspond to aggregation queries such that the input/output aggregates (small filled circles) are known by the adversary.

about aggregates in a chain. Hence, the target sequence must be "fresh", implying that one must be able to forge the underlying MAC.

The formal proof that the construction is indeed secure is given in the full version.

Acknowledgments. We thank all the anonymous reviewers for their comments. This work was supported by the Emmy Noether Program Fi 940/2-1 of the German Research Foundation (DFG).

References

1. Mihir Bellare. *New Proofs for NMAC and HMAC: Security without Collision Resistance.* Advances in Cryptology — Crypto'06, Lecture Notes in Computer Science, pages 602–619. Springer-Verlag, 2006.
2. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. *Aggregate and Verifiably Encrypted Signatures from Bilinear Maps.* Advances in Cryptology — Eurocrypt'03, Lecture Notes in Computer Science, pages 416–432. Springer-Verlag, 2003.
3. Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. *Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing.* ACM Conference on Computer and Communications Security, pages 276–285. ACM Press, 2007.
4. Tetsu Iwata and Kaoru Kurosawa. *OMAC: One-Key CBC MAC.* Fast Software Encryption (FSE)2003, Lecture Notes in Computer Science, pages 129–153. Springer-Verlag, 2003.
5. Jonathan Katz and Andrew Y. Lindell. *Aggregate Message Authentication Codes.* Topics in Cryptology — Cryptographer's Track, RSA Conference (CT-RSA)'08, Lecture Notes in Computer Science, pages 155–169. Springer-Verlag, 2008.
6. Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. *Sequential Aggregate Signatures from Trapdoor Permutations.* Advances in Cryptology — Eurocrypt'04, Lecture Notes in Computer Science, pages 74–90. Springer-Verlag, 2004.
7. Gregory Neven. *Efficient Sequential Aggregate Signed Data.* Advances in Cryptology — Eurocrypt'08, Lecture Notes in Computer Science, pages 52–69. Springer-Verlag, 2008.

A Mix-and-Match Attacks on the Katz-Lindell Scheme

In this section we discuss several attack strategies against the Katz-Lindell [5] aggregate MAC scheme. Note that our attacks do not contradict the security results in [5], because the scheme has only been designed to meet the relaxed unforgeability notion.

The aggregation step in the Katz-Lindell scheme is rather simple: the aggregation algorithm computes the exclusive-or over all (deterministically computed) tags resp. in our routing example of four nodes $N_1 \to N_2 \to N_3 \to N_4$, node N_i adds its MAC for message M_i to the current aggregate value.

Deletion Attack. Given the replies

$$\sigma_1 = \mathsf{Mac}(sk_1, M_1) \oplus \mathsf{Mac}(sk_2, M_2) \oplus \mathsf{Mac}(sk_3, M_3), \qquad \sigma_2 = \mathsf{Mac}(sk_2, M_2)$$

to two aggregation queries for message sets $\{M_1, M_2, M_3\}$, and $\{M_2\}$, where each message M_i is given to node N_i, the adversary is able to delete the element $\sigma_2 = \mathsf{Mac}(sk_2, M_2)$ from the aggregate:

$$\sigma_1^* = \sigma_1 \oplus \sigma_2 = \mathsf{Mac}(sk_1, M_1) \oplus \mathsf{Mac}(sk_3, M_3)$$

and obtains a valid "fresh" aggregate on the set (on the invalid route $N_1 \rightarrow N_3$) $\{M_1, M_3\}$.

Re-Ordering Attack. Given the replies

$$\sigma_1 = \mathsf{Mac}(sk_1, M_1) \oplus \mathsf{Mac}(sk_2, M_2), \quad \sigma_2 = \mathsf{Mac}(sk_1, M_3) \oplus \mathsf{Mac}(sk_2, M_2), \quad \sigma_3 = \mathsf{Mac}(sk_1, M_1) \oplus \mathsf{Mac}(sk_2, M_4)$$

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

$$\sigma^* = \sigma_1 \oplus \sigma_2 \oplus \sigma_3 = \mathsf{Mac}(sk_1, M_3) \oplus \mathsf{Mac}(sk_2, M_4)$$

for the set $\{M_3, M_4\}$.

Extension Attack. Given the replies

$$\sigma_1 = \mathsf{Mac}(sk_1, M_1) \oplus \mathsf{Mac}(sk_2, M_2) \oplus \mathsf{Mac}(sk_3, M_3) \; \sigma_2 = \mathsf{Mac}(sk_A, M_A) \oplus \mathsf{Mac}(sk_B, M_B)$$

to the aggregation queries for message sets $\{M_1, M_2, M_3\}$ and $\{M_A, M_B\}$, the adversary is able to extend the aggregate:

$$\sigma_3^* = \sigma_1 \oplus \sigma_2 = \mathsf{Mac}(sk_1, M_1) \oplus \mathsf{Mac}(sk_A, M_A) \oplus \mathsf{Mac}(sk_3, M_3) \oplus \mathsf{Mac}(sk_B, M_B) \oplus \mathsf{Mac}(sk_2, M_2)$$

and obtains a valid "fresh" aggregate on the set (on the route) $\{M_1, M_A, M_3, M_B, M_2\}$ (for arbitrary M_A, M_B).

Recombination Attack. Given the replies

$$\sigma_1 = \mathsf{Mac}(sk_1, M_1) \oplus \mathsf{Mac}(sk_2, M_2) \oplus \mathsf{Mac}(sk_3, M_3),$$
$$\sigma_2 = \mathsf{Mac}(sk_2, M_2) \oplus \mathsf{Mac}(sk_3, M_3) \oplus \mathsf{Mac}(sk_4, M_4),$$

and

$$\sigma_3 = \mathsf{Mac}(sk_3, M_3) \oplus \mathsf{Mac}(sk_4, M_4) \oplus \mathsf{Mac}(sk_5, M_5),$$

to the aggregation queries for message sets $\{M_1, M_2, M_3\}$, $\{M_2, M_3, M_4\}$, and $\{M_3, M_4, M_5\}$, the adversary is able to recombine the aggregate,

$$\sigma^* = \sigma_1 \oplus \sigma_2 \oplus \sigma_3 = \mathsf{Mac}(sk_1, M_1) \oplus \mathsf{Mac}(sk_5, M_5) \oplus \mathsf{Mac}(sk_3, M_3),$$

and obtains a valid "fresh" aggregate on the set (on the route) $\{M_1, M_5, M_3\}$.

Note also that, if we assume N_i only accepts input from N_{i-1} then replay attacks do not necessarily help, because the adversary can never send a previously obtained tuple $\{M_1, M_2\}$ to the node N_4.

B Preliminaries: MACs and Their Security

Definition 12 (Message Authentication Codes). *A message authentication code scheme* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ *is a triple of efficient algorithms where*

Key Generation. $\mathsf{KGen}(1^n)$ *gets as input the security parameter* 1^n *and returns a key* sk.

Authentication. *The authentication algorithm* $\sigma \leftarrow \mathsf{Mac}(sk, m)$ *takes as input the key* sk*, a message* m *from a message space* $\mathcal{M}_n$ *and returns a tag* σ *in a range* $\mathcal{R}_n$.

Verification. $\mathsf{Vf}(sk, m, \sigma)$ *returns a bit.*

It is assumed that the scheme is complete*, i.e., for all* $sk \leftarrow \mathsf{KGen}(1^n)$*, any* $m \in \mathcal{M}_n$*, and any* $\sigma \leftarrow \mathsf{Mac}(sk, m)$ *we have* $\mathsf{Vf}(sk, m, \sigma) = 1$.

A message authentication code is called *deterministic* if the tagging algorithm is deterministic. A deterministic MAC is called *canonical* if the verification algorithm recomputes the tag for a given message and checks that it matches the given one. Unforgeability demands that it is infeasible to produce a valid tag for a new message:

Definition 13 (Unforgeability). *A message authentication code* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf})$ *is* (t, q_t, q_v, ϵ)-unforgeable under chosen message attacks *(EU-CMA) if for any algorithm* $\mathcal{A}$ *running in time* t *the probability that the experiment* $\mathsf{Forge}_{\mathcal{A}}^{\mathsf{MAC}}$ *evaluates to* 1 *is at most* $\epsilon(n)$*, where*

Experiment $\mathsf{Forge}_{\mathcal{A}}^{\mathsf{MAC}}(n)$
$sk \leftarrow \mathsf{KGen}(1^n)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Mac}(sk,\cdot),\mathsf{Vf}(sk,\cdot,\cdot)}(1^n)$
Return 1 *if, at some point,* $\mathcal{A}$ *makes a query* m^*, σ^* *to* Vf *such that* $\mathsf{Vf}(sk, m^*, \sigma^*) = 1$ *and* $\mathcal{A}$ *has never queried* $\mathsf{Mac}(sk, \cdot)$ *about* m^* *before.*

and $\mathcal{A}$ *makes at most* q_t *queries to oracle* Mac *and at most* q_v *queries to oracle* Vf*. The probability is taken over all coin tosses of* $\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf}$ *and* $\mathcal{A}$.

C Counter-Based Aggregation-Unforgeable Schemes

In this section we show that, assuming the existence of a shared counter, we can lift non-ordered aggregate schemes that are unforgeable in the classical sense to achieve a stronger security requirement. However, even such an assumption only allows to prevent some of the mix-and-matching attacks discussed in the introduction, but not attacks aiming to erase subsets of previously queried aggregates. Thus, we introduce a slightly weaker definition of aggregation-unforgeability by considering forgeries that consist of those subsets as trivial. This corresponds to the case that the adversary can remove contributions of honest parties from valid aggregates. Recall that the adversary can query an aggregation oracle, denoted by OAgg, which takes the key/identity pairs (sk_i, id_i) of all honest parties (provided by the system) and a set of message/identity pairs $M = \{(m_1, \mathsf{id}_1), \ldots, (m_k, \mathsf{id}_k)\}$ (chosen by the adversary). The oracle returns an aggregate MAC σ for these data.

To mark subsets of aggregation queries as trivial, we include all message sets that are subsets of queried aggregates into the closure:

$$\mathsf{Closure}^*(Q_{\mathsf{Agg}}, Q_{\mathsf{Cor}}) = \left\{ \bigcup_{M_A \in A} M_A \cup \bigcup_{M_A^* \subseteq A^*} M_A^* \cup M_C \;\middle|\; A^* \in Q_{\mathsf{Agg}}, A \subseteq Q_{\mathsf{Agg}}, M_C \subseteq \bigcup_{\mathsf{id}^* \in Q_{\mathsf{Cor}}} \{(m, \mathsf{id}^*) \mid m \in \mathcal{M}_n\} \right\}.$$

We now show how to construct an aggregate tag scheme that is aggregation-unforgeable with respect to the weaker definition.

Given an aggregate MAC scheme $\mathsf{Agg} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vf}, \mathsf{Agg}, \mathsf{AVf})$ that is unforgeable according to the definition of Katz and Lindell [5] we can derive a counter-based aggregate tag scheme that achieves our stronger security requirement of aggregation-unforgeability. To this end, we augment the aggregate tag scheme as follows:

- KGen and AVf remain unchanged
- $\mathsf{Mac}^*(sk, m)$ queries Mac on the string $m^* = (\mathsf{count}, m)$ where count is a synchronized counter shared between all signing parties. It outputs $\sigma^* = (\mathsf{count}, \sigma)$ with $\sigma \leftarrow \mathsf{Mac}(sk, (\mathsf{count}, m))$ and updates the counter.
- $\mathsf{Agg}^*(M, \sigma)$ parses σ as $\{(\mathsf{count}_1, \sigma_1), \ldots, (\mathsf{count}_\ell, \sigma_\ell)\}$ and stops with output $\perp$ if the counter values differ. Else, it computes $\sigma \leftarrow \mathsf{Agg}(M^*, \{\sigma_1, \ldots, \sigma_\ell\})$ for $m_i^* = (\mathsf{count}, m_i)$ and outputs $\sigma^* = (\mathsf{count}, \sigma)$.
- $\mathsf{AggVf}^*(\mathsf{sk}, M, (\mathsf{count}, \sigma))$ sets $M^* = \{(\mathsf{count}, m_1), \ldots, (\mathsf{count}, m_\ell)\}$ for $M = \{m_1, \ldots, m_\ell\}$ and outputs $\mathsf{AggVf}(\mathsf{sk}, M^*, \sigma)$.

Prepending a unique counter value to the messages in each signing request prevents the adversary from recombining several aggregates into a new one, as the verification algorithm first checks that all messages carry the same counter value. Recall that the strength of the adversary in our security model stems from granting an aggregation oracle and considering non-trivial recombinations of aggregates as valid forgeries. However, if the adversary tries to exploit the potential of recombining aggregates into fresh ones, he has to ensure that all counter values are equal. Hence, the adversary can at most delete messages from aggregates or add values by corrupt parties.

More formally, assume that the adversary eventually outputs a valid forgery for message set M and tag (count, σ). Suppose that there is at least one honest party in the corresponding set (else the attempt is trivially in the closure). If the set of augmented messages (count, m_i) contains a value previously not tagged by an honest party, then the security follows from the (basic) unforgeability notion of the underlying scheme.

Hence, suppose all pairs (count, m_i) for honest parties have been tagged before. Since each counter value is used only once, there is a unique aggregation query where tags for these pairs have been computed. It follows that the forgery attempt only contains a subset of this query (and possibly additional contributions by corrupt players). But then the attempt is in the closure.