Aggregate Message Authentication Codes

Jonathan Katz
Dept. of Computer Science
University of Maryland, USA
jkatz@cs.umd.edu

Yehuda Lindell
Dept. of Computer Science
Bar-Ilan University, Israel
lindell@cs.biu.ac.il

Abstract

We propose and investigate the notion of *aggregate message authentication codes (MACs)* which have the property that multiple MAC tags, computed by (possibly) different senders on multiple (possibly different) messages, can be aggregated into a shorter tag that can still be verified by a recipient who shares a distinct key with each sender. We suggest aggregate MACs as an appropriate tool for authenticated communication in mobile ad-hoc networks or other settings where resource-constrained devices share distinct keys with a single entity (such as a base station), and communication is an expensive resource.

1 Introduction

Aggregate signatures, introduced by Boneh et al. [5, 14], allow t distinct signatures by t (possibly different) signers on t (possibly different) messages to be *aggregated* into a shorter signature that still suffices to convince a verifier that each signer did indeed sign the appropriate message. Since their introduction, various aggregate signature schemes have been proposed [12, 11, 8, 4]. No formal attention has yet been dedicated to the *private-key* analogue of aggregate signatures: aggregate message authentication codes (MACs). In this paper, we initiate a formal study of this primitive.

One reason for the relative lack of attention focused on aggregate MACs may be the (incorrect) perception that they are of limited value. Indeed, the applications suggested in [5] — such as compressing certificate chains, or reducing the message size in secure routing protocols — are all specific to the public-key (rather than the private-key) setting. Nevertheless, we suggest that aggregate MACs *can* be useful in specific domains. As perhaps the most compelling example, consider the problem of authenticated communication in a mobile ad-hoc network (MANET), where communication is considered an "expensive" resource because of its effect on the battery life of the nodes. Here, there is a collection of t nodes $U_1, \ldots, U_t$, each of whom is interested in sending messages to a base station B. We assume that the base station shares in advance a key k_i with each node U_i, and that node U_i authenticates any outgoing message m_i by computing $\mathsf{tag}_i = \mathsf{Mac}_{k_i}(m_i)$.

Most nodes cannot communicate directly with the base station due to the limited range of their wireless devices, and so all communication is instead routed among the nodes themselves until it reaches the base station. For simplicity in this example, let us assume that nodes are arranged in a (logical) binary tree so that each node U_i at a leaf sends (m_i, tag_i) to its parent, and each internal node U_j forwards to its own parent all the communication from its children in addition to (m_j, tag_j). The root U^* in this example is the only node that is able to communicate directly with the base station, and it forwards to the base station the communication from all nodes in the network along with its own contribution (m^*, tag^*).

The messages themselves may be very short — corresponding, e.g., to temperature readings or even just an indicator bit. For concreteness, say messages are 16 bits long. (Replay attacks can be addressed by using a counter shared by the base station and all the nodes; this counter would be authenticated by each node along with the message, but would not need to be transmitted and so does not affect the communication complexity in the calculation that follows.) Furthermore, assume the length of a MAC tag is 160 bits (e.g., if HMAC is used), and take $t = 10^4$. The communication from the root node alone to the base station is then $(160+16) \cdot t = 1.76 \times 10^6$ bits, while the total communication in the network is (roughly) $(160 + 16) \cdot (t \log t) \approx 2.3 \times 10^7$ bits.

The above description assumes MACs used in the standard fashion, meaning that all MAC tags are transmitted together with the messages. If an aggregate MAC were available, however, then each node U_j would be able to *combine* its own MAC tag with those of its children. Say this aggregation can be performed while maintaining the MAC tag length, even of aggregated tags, at 160 bits. (Our construction will achieve this.) The communication from the root to the base station will now be only $160 + 16t \approx 1.6 \times 10^5$ bits, and the total communication in the network will be improved to roughly $16(t \log t) + 160t \approx 3.7 \times 10^6$ bits, for roughly an order of magnitude improvement in each case.

Aggregate MACs could also be used to improve the communication complexity in schemes such as those of [13] or [10] which deal with aggregation of *data*. We do not explore this further here, as we view the use of such techniques as tangential to the main thrust of this paper.

1.1 Our Contributions

Motivated in part by scenarios such as the above, we formally introduce the notion of aggregate MACs and initiate the first detailed study of this primitive. After giving appropriate definitions, we show a simple and highly efficient construction of aggregate MACs based on a wide variety of existing (standard) MACs. The existence of *efficient* aggregate MACs is somewhat surprising since, in the setting of aggregate signatures, algebraic (i.e., number-theoretic) properties of the underlying signature scheme are used to perform aggregation. In contrast, here we would like to avoid number-theoretic constructions and base aggregate MACs on primitives like block ciphers and hash functions that have limited algebraic structure.

Summarizing, we prove the following (informally stated) theorem:

Theorem *If there exists a secure message authentication code, then there exists a secure* **aggregate** *message authentication code with complexity as follows:*

- **Aggregate MAC tag length:** *the same as for the basic MAC scheme*
- **Computation of a MAC tag in the aggregate scheme:** *the same as for the basic MAC scheme*
- **Computation of MAC tag aggregation:** *linear in the number of tags to be aggregated*
- **Verification of aggregated MAC tags:** *linear in the number of tags to be verified.*

As can be seen from above, the complexity of our aggregate construction is essentially the same as for a regular MAC scheme. This may be somewhat surprising since in the public-key setting of aggregate signatures, it is significantly harder to obtain secure aggregation. The reason for this will become clear after seeing our construction in Section 3.

Verifying individual messages. Our aggregate scheme works very well when the receiver wishes to verify the authenticity of all the aggregated messages. However, if the receiver wishes to verify only one or a few messages, it must still verify them all. In Section 4, we explore a variant of our main construction that offers a trade-off between the length of an aggregate tag and the time required to verify individual messages. We also prove a lower bound showing that if constant or logarithmic-time verification of individual messages is desired, then the aggregated tag length must be linear in the total number of messages whose tags are aggregated (and so the trivial approach of concatenating individual tags is optimal up to a multiplicative factor).

1.2 Related Work

Subsequent to our work, we became aware of two other recent papers [6, 3] that, *inter alia*, use what are essentially aggregate MACs (and, in fact, use essentially the same construction we show in Section 3). The key additional contributions of our work are: (1) we provide a formal definition of the problem and a proof of security for our construction; (2) we suggest extensions of the construction offering the time/length trade-off discussed above; and (3) we show a lower bound on the required tag length when fast verification of individual messages is required.

Following our work, Eikemeier et al. [7] have explored the notion of *history freeness* for aggregate MACs.

2 Definitions

Our definitions are based on those given in [5, 14] for aggregate signatures. Rather than exploring numerous possible special cases of the definitions, we make our definitions as general as possible (and our construction will achieve these definitions). We begin with a functional definition. The security parameter, which determines the length of the key, will be denoted by n.

Definition 1 An aggregate message-authentication code (MAC) is a tuple of probabilistic polynomial-time algorithms $(\mathsf{Mac}, \mathsf{Agg}, \mathsf{Vrfy})$ such that:

- **Authentication algorithm** Mac**:** upon input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^*$, algorithm Mac outputs a *tag* tag. We denote this procedure by $\mathsf{tag} \leftarrow \mathsf{Mac}_k(m)$.
- **Aggregation algorithm** Agg**:** upon input two sets of message/identifier[1] pairs $M^1 = \{(m_1^1, \mathsf{id}_1^1), \ldots, (m_{\ell_1}^1, \mathsf{id}_{\ell_1}^1\}$, $M^2 = \{(m_1^2, \mathsf{id}_1^2), \ldots, (m_{\ell_2}^2, \mathsf{id}_{\ell_2}^2)\}$ and associated tags $\mathsf{tag}^1, \mathsf{tag}^2$, algorithm Agg outputs a new tag tag. We stress that this algorithm is unkeyed.
- **Verification algorithm** Vrfy**:** upon receiving a set of key/identifier pairs $\{(k_1, \mathsf{id}_1), \ldots, (k_t, \mathsf{id}_t)\}$, a set of message/identifier pairs $M = \{(m_1, \mathsf{id}'_1), \ldots, (m_\ell, \mathsf{id}'_\ell)\}$, and a tag tag, algorithm Vrfy outputs a single bit, with '1' denoting acceptance and '0' denoting rejection. We denote this by $\mathsf{Vrfy}_{(k_1,\mathsf{id}_1),\ldots,(k_n,\mathsf{id}_t)}(M, \mathsf{tag})$. (In normal usage, $\mathsf{id}'_i \in \{\mathsf{id}_1, \ldots, \mathsf{id}_t\}$ for all i.)

The following correctness conditions are required to hold:

- For all $k, \mathsf{id}, m \in \{0,1\}^*$, it holds that $\mathsf{Vrfy}_{k,\mathsf{id}}((m, \mathsf{id}), \mathsf{Mac}_k(m)) = 1$. (This is essentially the correctness condition for standard MACs.)
- Let M^1, M^2 be sets of message/identifier pairs with[2] $M^1 \cap M^2 = \emptyset$. If:

[1] We discuss the role of the identifiers below.

[2] This technical condition ensures that the same message/identifier pair does not appear in both M^1 and M^2.

1. $\mathsf{Vrfy}_{(k_1,\mathsf{id}_1),\ldots,(k_t,\mathsf{id}_t)}(M^1,\mathsf{tag}^1)=1$, and
2. $\mathsf{Vrfy}_{(k_1,\mathsf{id}_1),\ldots,(k_t,\mathsf{id}_t)}(M^2,\mathsf{tag}^2)=1$,

then $\mathsf{Vrfy}_{(k_1,\mathsf{id}_1),\ldots,(k_n,\mathsf{id}_n)}(M,\mathsf{Agg}(M^1,M^2,\mathsf{tag}^1,\mathsf{tag}^2))=1$, where $M=M^1\cup M^2$. Thus, aggregation of MAC tags preserves correct verification.

■

The use of identifiers provides a way to differentiate between different senders; in order to know which secret key to use for verification, the receiver needs to know which message is associated with which sender. (Thus enforcing $M^1\cap M^2=\emptyset$ in the second correctness condition just means that aggregation is not applied if the same sender authenticated the same message twice.) Identifiers are not needed in the setting of aggregate signatures where each sender is associated with a unique public key which, in effect, serves as an identifier. For simplicity in what follows, we write $\mathsf{Vrfy}_{k_1,\ldots,k_t}(\cdot,\cdot)$ for the verification algorithm, and we sometimes find it convenient to set $\mathsf{id}_i=i$ (this has no effect on our results).

An aggregate MAC can be used as follows. A receiver R who wants to receive authenticated messages from t senders begins by sharing uniform keys $k_1,\ldots,k_t\in\{0,1\}^n$ with each sender (i.e., key k_i is shared with the sender with identity id_i). When sender id_i wishes to authenticate a message m_i, it simply computes $\mathsf{tag}^i\leftarrow\mathsf{Mac}_{k_i}(m_i)$. Given a tag computed in this way, and a second tag tag^j computed by sender id_j on the message m_j, these two tags can be aggregated by computing the value $\mathsf{tag}\leftarrow\mathsf{Agg}(\{(m_i,\mathsf{id}_i)\},\{(m_j,\mathsf{id}_j)\},\mathsf{tag}^i,\mathsf{tag}^j)$. Given the result tag, the receiver can check that sender id_i authenticated m_i and that sender id_j authenticated m_j by computing

$$\mathsf{Vrfy}_{k_i,k_j}\left(\{(m_i,\mathsf{id}_i),(m_j,\mathsf{id}_j)\},\mathsf{tag}\right)$$

and verifying that the output is 1. Note that we do not assume $\mathsf{id}_i\neq\mathsf{id}_j$ but, as per footnote 2, we do assume $(m_i,\mathsf{id}_i)\neq(m_j,\mathsf{id}_j)$.

As in the case of aggregate signatures, our definition of security corresponds to existential unforgeability under an adaptive chosen-message attack [9]. Because we are in the shared-key setting, however, there are some technical differences between our definition and the security definition for aggregate signatures. In particular, we consider an adversary who may adaptively corrupt various senders and learn their secret keys, and require security to hold also in such a setting.

Definition 2 Let $\mathcal{A}$ be an adversary, and consider the following experiment parameterized by a security parameter n:

- **Key generation:** Keys $k_1,\ldots,k_t\in\{0,1\}^n$ are generated.
- **Attack phase:** $\mathcal{A}$ may query the following oracles:
  - **Message-authentication oracle** Mac**:** On input (i,m), the oracle returns $\mathsf{Mac}_{k_i}(m)$.
  - **Corruption oracle** $\mathsf{Corrupt}$**:** upon input i, the oracle returns k_i.
- **Output:** The adversary $\mathcal{A}$ outputs a set of message/identifier pairs $M=\{(m_1,\mathsf{id}_1),\ldots,(m_\ell,\mathsf{id}_\ell)\}$ and a tag tag. All the pairs in M are required to be distinct.
- **Success determination:** We say $\mathcal{A}$ $\mathsf{succeeds}$ if (1) $\mathsf{Vrfy}_{k_1,\ldots,k_t}(M,\mathsf{tag})=1$ and (2) there exists a pair $(m_{i^*},id_{i^*})\in M$ such that

1. $\mathcal{A}$ never queried $\mathsf{Corrupt}(\mathsf{id}_{i^*})$, and
2. $\mathcal{A}$ never queried $\mathsf{Mac}(\mathsf{id}_{i^*}, m_{i^*})$.

Aggregate MAC $(\mathsf{Mac}, \mathsf{Agg}, \mathsf{Vrfy})$ is secure if for all $t = \mathsf{poly}(n)$ and all probabilistic polynomial-time adversaries $\mathcal{A}$, the probability that $\mathcal{A}$ succeeds in the above experiment is negligible. ■

We do not consider verification queries even though, in general, they may give the adversary additional power [1]. This is justified by the fact that our eventual construction satisfies the conditions stated in [1] for which verification queries do *not* give any additional power. (Of course, they prove this only for standard MACs but it is easy to see that their proof carries over to our setting as well.) Note also that we need not allow "aggregate" queries in the above definition, since the aggregation algorithm Agg is unkeyed.

3 Constructing Aggregate MACs

In this section, we show that aggregate MACs can be constructed from essentially any standard message authentication code. We begin by illustrating the idea using as a building block the simple (standard) message authentication code constructed from a pseudorandom function F with output length n as follows: $\mathsf{Mac}_k(m) = F_k(m)$. In this case, given tags $\mathsf{tag}_1, \ldots, \mathsf{tag}_\ell$ associated with message/identifier pairs (m_i, i), respectively, we can aggregate these tags by simply computing the XOR of all the tag values; i.e.,

$$\mathsf{tag} = \mathsf{tag}_1 \oplus \mathsf{tag}_2 \oplus \cdots \oplus \mathsf{tag}_\ell.$$

(For simplicity, we consider identifiers $1, \ldots, \ell$ above. However, as we will see in the formal description below, these identifiers need not be distinct.) Verification is carried out in the obvious way: given a set of message/identifier pairs $M = \{(m_1, 1), \ldots, (m_\ell, \ell)\}$ and tag, the receiver outputs 1 if and only if

$$\mathsf{tag} = \bigoplus_{i=1}^{\ell} F_{k_i}(m_i).$$

As for the security of this scheme, we may argue informally as follows: say an adversary outputs $\{(m_1, \mathsf{id}'_1), \ldots, (m_\ell, \mathsf{id}'_\ell)\}$ and tag such that there exists an i for which $\mathcal{A}$ did not query either $\mathsf{Corrupt}(\mathsf{id}'_i)$ or $\mathsf{Mac}(\mathsf{id}'_i, m_i)$. Let $i^* = \mathsf{id}'_i$. Then, from the point of view of the adversary, the value $F_{k_{i^*}}(m_i)$ looks random. Since XORing a random(-looking) value with any other (uncorrelated) strings yields a random(-looking) string, we see that the value $\bigoplus_{i=1}^{\ell} F_{k_{\mathsf{id}'_i}}(m_i)$ computed by the receiver also looks random to the adversary, and cannot be guessed by the adversary with probability much better than 2^{-n}. We conclude that tag is a valid forgery with probability only negligibly better than 2^{-n}, and so the adversary cannot output a valid forgery except with negligible probability.

Extending the above ideas, we may realize that the proof does not require the individual MAC tag $F_{k_{i^*}}(m_i)$ to be *pseudorandom*, but instead only requires that it be *unpredictable*. But this holds for *any* secure (standard) MAC, by the definition of security for MACs. Thus, as far as security is concerned, the above approach works for *any* underlying MAC. On the other hand, verification in the aggregate MAC requires that verification in the underlying MAC be done by re-computing the MAC tag and checking equality with what is received. (I.e., $\mathsf{Vrfy}_k(m, \mathsf{tag})$ outputs 1 if and only if $\mathsf{Mac}_k(m) = \mathsf{tag}$.) We may assume, without loss of generality, that verification is done

this way for any *deterministic* MAC; for randomized MACs, however, verification *cannot* be done this way. This means that certain MACs (e.g., XOR-MAC [2]) cannot be utilized directly in the above construction, though we remark that any randomized MAC can be "derandomized" using a pseudorandom function. In any case, most commonly-used MACs are deterministic, and thus the restriction to deterministic MACs is not a serious one.

We now describe our aggregate MAC scheme formally, and rigorously prove its security with respect to Definition 2. Let $(\mathsf{Mac}, \mathsf{Vrfy})$ denote a standard message-authentication code where Mac is *deterministic*. (We will ignore the Vrfy algorithm from now on since, as noted above, we can perform verification by simply re-running Mac.) We have the following construction:

Construction 1 (Aggregate MAC Scheme)
Let Mac *be a deterministic algorithm. We define* $(\mathsf{Mac}^*, \mathsf{Agg}^*, \mathsf{Vrfy}^*)$ *as follows:*

- **Algorithm** Mac^***:** *given as input* $k \in \{0,1\}^n$ *and* $m \in \{0,1\}^*$*, output* $\mathsf{Mac}_k(m)$.
- **Algorithm** Agg^***:** *given as input two sets* M^1, M^2 *of message/identifier pairs and two tags* $\mathsf{tag}^1, \mathsf{tag}^2$*, output* $\mathsf{tag} = \mathsf{tag}^1 \oplus \mathsf{tag}^2$.
- **Algorithm** Vrfy^***:** *given as input keys* $k_1, \ldots, k_t \in \{0,1\}^n$ *and a set* $M = \{(m_1, i_1), \ldots, (m_\ell, i_\ell)\}$ *of message/identifier pairs where* $i_\ell \in \{1, \ldots, t\}$ *for all* ℓ*, compute the value* $\mathsf{tag}' = \bigoplus_{j=1}^{\ell} \mathsf{Mac}_{k_{i_j}}(m_j)$ *and output* 1 *if and only if* $\mathsf{tag}' = \mathsf{tag}$*. (We stress that the input to* Vrfy^* *is a* set*, and so all the tuples in* M *are distinct.)*

It is easy to verify correctness of the above scheme. As for security, we have:

Theorem 1 *If* $(\mathsf{Mac}, \mathsf{Vrfy})$ *is existentially unforgeable under an adaptive chosen-message attack, then* $(\mathsf{Mac}^*, \mathsf{Agg}^*, \mathsf{Vrfy}^*)$ *given in Construction* 1 *is a secure aggregate message authentication code.*

Proof: Fix a probabilistic polynomial-time adversary $\mathcal{A}$ and some $t = \mathsf{poly}(n)$ as in Definition 2. We construct a probabilistic polynomial-time algorithm $\mathcal{F}$ that interacts with an instance of $(\mathsf{Mac}, \mathsf{Vrfy})$ and attempts to produce a valid forgery for a previously unauthenticated message. $\mathcal{F}$ is given access to an oracle $\mathsf{Mac}_{k^*}(\cdot)$ for an unknown key k^*, and proceeds as follows:

1. It chooses a random $i^* \leftarrow \{1, \ldots, t\}$.
2. For $i = 1$ to t:
   (a) If $i \neq i^*$, choose $k_i \leftarrow \{0,1\}^n$.
   (b) If $i = i^*$, do nothing (however, *implicitly* set $k_{i^*} = k^*$).
3. Run $\mathcal{A}(1^n)$, answering its queries as follows:

   Query $\mathsf{Mac}^*(i, m)$**:** If $i \neq i^*$ then $\mathcal{F}$ answers the query using the known key k_i. If $i = i^*$ then $\mathcal{F}$ queries its own MAC oracle $\mathsf{Mac}_{k^*}(\cdot)$ and returns the result.

   Query $\mathsf{Corrupt}(i)$**:** If $i \neq i^*$ then give $\mathcal{A}$ the known key k_i. If $i = i^*$ then abort.
4. At some point, $\mathcal{A}$ outputs $M = \{(m_1, \mathsf{id}'_1), \ldots, (m_\ell, \mathsf{id}'_\ell)\}$ and tag. Let j be the first index such that (1) $\mathcal{A}$ never queried $\mathsf{Corrupt}(\mathsf{id}'_j)$ and (2) $\mathcal{A}$ never queried $\mathsf{Mac}^*(\mathsf{id}'_j, m_j)$. (We assume without loss of generality that some such j exists.) If $\mathsf{id}'_j \neq i^*$ then abort; otherwise, proceed as described below.

5. Assuming $\mathsf{id}'_j = i^*$, algorithm $\mathcal{F}$ computes

$$\mathsf{tag}^* = \mathsf{tag} \oplus \left(\bigoplus_{i \neq j} \mathsf{Mac}_{k_{\mathsf{id}'_i}}(m_i) \right) ,$$

where $\mathcal{F}$ computes $\mathsf{Mac}_{k_{\mathsf{id}'_i}}(m_i)$ using the known key $k_{\mathsf{id}'_i}$ when $\mathsf{id}'_i \neq i^*$, and computes $\mathsf{Mac}_{\mathsf{id}'_i}(m_i)$ by querying its MAC oracle $\mathsf{Mac}_{k^*}(\cdot)$ when $\mathsf{id}'_i = i^*$. Finally, $\mathcal{F}$ outputs (m_j, tag^*).

The proof follows easily from the following observations:

- The probability that $\mathcal{F}$ does not abort is exactly $1/t$, which is inverse polynomial. Furthermore, conditioned on not aborting, the simulation that $\mathcal{F}$ provides for $\mathcal{A}$ is perfect.
- If $\mathcal{A}$ succeeds in a given execution (and $\mathcal{F}$ does not abort), then $\mathcal{F}$ outputs a valid forgery. To see this, note that when $\mathcal{A}$ succeeds this means that

$$\bigoplus_{i=1}^{\ell} \mathsf{Mac}_{k_{\mathsf{id}'_i}}(m_i) = \mathsf{tag},$$

where we stress that $\mathsf{Mac}_{k_{\mathsf{id}'_i}}(m_i)$ is a *fixed*, well-defined value by virtue of the fact that Mac is deterministic. Thus, the value tag^* output by $\mathcal{F}$ is equal to the (well-defined) value $\mathsf{Mac}_{k_{i^*}}(m_j) = \mathsf{Mac}_{k^*}(m_j)$. Furthermore, $\mathcal{F}$ has never queried its own MAC oracle with the message m_j since, by assumption, $\mathcal{A}$ never queried $\mathsf{Mac}^*(i^*, m_j)$ prior to step 5 of $\mathcal{F}$'s execution, above, and $\mathcal{F}$ will not query m_j to its MAC oracle in step 5 since all tuples in the set M are distinct.

This completes the proof. ■

Efficiency. Our construction of aggregate MACs is highly efficient. Consider the example of a mobile ad-hoc network (MANET) as described in the introduction. If the nodes are arranged in a tree, then each node receives a set of messages together with a single tag from each of its children. In order to forward the messages, all the node needs to do is to concatenate the lists of messages, compute its own MAC, and XOR all the tags together.

4 An Extension and a Lower Bound

A limitation of the construction given in the previous section is that the receiver must re-compute the (individual) MAC tags on *all* ℓ messages whose tags have been aggregated. This is not a limitation in the MANET example given above. However, in some cases, the receiver may only be interested in verifying the authenticity of a *single* message (or some small subset of the messages). In such cases, the requirement to re-compute the MAC tags of all the messages is undesirable.

In this section, we present a simple idea that offers a trade-off between the length of the aggregate tag and the time required to verify integrity of a single message. To achieve authentication of a single message in *constant* time (i.e., a constant number of calls to the algorithms of the underlying MAC scheme), independent of the number of aggregated tags ℓ, our approach yields a tag of length

$\mathcal{O}(\ell \cdot T)$, where T is the length of the tag in the underlying MAC. This is not of much interest because we can achieve a tag of length $\mathcal{O}(\ell \cdot T)$ by just concatenating the tags of a standard MAC (i.e., aggregation is just concatenation). However, our approach yields a general tradeoff where the product of the authentication time and tag length is $\mathcal{O}(\ell \cdot T)$. At one extreme, we showed in the previous section how to achieve authentication in time ℓ with a tag of length T. At the other extreme, concatenating MAC tags gives authentication in constant time but has a tag of length $\ell \cdot T$. Our approach, described below, allows essentially anything in between. In particular, one can achieve authentication in time $\mathcal{O}(\sqrt{\ell})$ with a tag of length $\mathcal{O}(\sqrt{\ell} \cdot T)$.

It is interesting to wonder whether this is optimal. In this direction, we also present a lower bound showing that this approach *is* asymptotically optimal (up to a multiplicative factor of T) when considering verification that takes constant, or at most logarithmic, time. That is, we show that any aggregate MAC scheme that enables authentication in time $O(\log \ell)$ must have a tag of length at least $\Omega(\ell)$.

4.1 The Construction

Before presenting our construction, we first describe the problem in a bit more detail. Recall from Definition 1 that the receiver holds a set of keys $k_1, \ldots, k_t$, and is assumed to receive a set of message/identifier pairs $M = \{(m_1, \mathsf{id}_1), \ldots, (m_\ell, \mathsf{id}_\ell)\}$ and a tag tag. In this section, we assume the receiver does not care to simultaneously verify the authenticity of *all* messages in M (with respect to the identifier associated with each message) as in the previous section, but instead is interested only in verifying authenticity of *one* of the messages m_i (with respect to the associated identifier id_i).

A fairly straightforward solution is as follows. Fix some parameter ℓ'. Then run multiple instances of the "base aggregation scheme" from the previous section in parallel, but only aggregating at most ℓ' messages/tags using any given instance. (We stress that each sender still holds only one key, the verifier still holds one key per sender, and the Mac^* algorithm is unchanged. All that changes is the way aggregation and verification are performed.) The net result is that a set of message/identifier pairs $M = \{(m_1, \mathsf{id}_1), \ldots, (m_\ell, \mathsf{id}_\ell)\}$ is now authenticated by a sequence of $\ell^* = \lceil \ell/\ell' \rceil$ tags $\mathsf{tag}_1, \ldots, \mathsf{tag}_{\ell^*}$ generated according to the base aggregate MAC described previously, where tag_1 authenticates $m_1, \ldots, m_{\ell'}$ (with respect to the appropriate associated identities), tag_2 authenticates $m_{\ell'+1}, \ldots, m_{2\ell'}$, etc. To verify the authenticity of any particular message m_i, the verifier need only re-compute MAC tags for (at most) ℓ' messages in total.

The tag when ℓ messages are authenticated is now the length of $\lceil \ell/\ell' \rceil$ basic MAC tags (i.e., length $\lceil \ell/\ell' \rceil \cdot T$), and the time for verifying any particular message is improved to $\mathcal{O}(\ell')$ (instead of $\mathcal{O}(\ell)$ as previously). Thus, for example, setting $\ell' = \sqrt{\ell}$ we obtain verification of time $\mathcal{O}(\sqrt{\ell})$ and a tag that is comprised of $\sqrt{\ell}$ basic MAC tags. We remark that the time required to verify *all* the messages is essentially the same as before. Achieving *constant* verification time for any single message using this approach would result in a tag of (total) length linear in the number of messages being authenticated. In particular, when $\ell' = 1$ we obtain an "aggregate" scheme which simply concatenates MAC tags of all the messages being authenticated.

4.2 A Lower Bound

As we have mentioned, when constant verification time is desired (i.e., $\ell' = 1$ in the scheme of the previous section), the result is a MAC tag that consists of ℓ basic MAC tags (i.e., the aggregation

works by just concatenating MAC tags). This is rather disappointing and it would be highly desirable to improve this situation. In this section we show that the scheme presented above is essentially optimal. Specifically, we show that if verification can be carried out in constant time (or even in time $\mathcal{O}(\log \ell)$), then the aggregated tag must be at least $\Omega(\ell)$ bits long.

Before proceeding further, we observe this does not contradict the positive result obtained above. This is because we must have $T = \omega(\log n)$ (otherwise an adversary can guess a valid MAC tag, in the underlying scheme, with non-negligible probability) and because ℓ, the number of aggregated MACs, can be at most polynomial in n (or else it does not make much sense to talk about security of the scheme). Thus, the tag length of our previous construction when $\ell' = \mathcal{O}(\log \ell)$ is

$$T \cdot \ell / O(\log \ell) = \omega(\log n) \cdot \ell / \mathcal{O}(\log n) = \omega(\ell),$$

as required. We now formally state and prove the lower bound:

Theorem 2 *Fix a secure aggregate MAC* $(\mathsf{Mac}, \mathsf{Agg}, \mathsf{Vrfy})$ *with perfect correctness, and fix some* $\ell = \mathsf{poly}(n) \geq 2$. *Assume that, given an aggregate tag on some set of* ℓ *messages, verification of any single message can be done by reading at most* $O(\log \ell)$ *of the messages. Then an aggregate tag on* ℓ *messages has length* $\Omega(\ell)$.

Proof: Let T denote the length of the aggregate tag on ℓ messages. We show how the aggregate MAC can be used to transmit an ℓ-bit message from one party to another, with low (constant) probability of error, by sending only T bits. It follows that $T = \Omega(\ell)$.

Let A and B denote two parties who have shared a uniform $k \in \{0,1\}^n$ in advance. A is then given a uniform message $x \in \{0,1\}^\ell$ that it wants to communicate to B. To do so, A and B proceed as follows:

- A computes a message and sends it to B as described next.

  1. Party A encodes its input $x = x_1 \cdots x_\ell$ as a set of ℓ messages $M = \{m_1, \ldots, m_\ell\}$ by setting $m_i = \langle i \rangle \| x_i$ for every i, where $\langle i \rangle$ denotes a fixed-length binary encoding of i.
  2. A computes $\mathsf{tag}_i \leftarrow \mathsf{Mac}_k(m_i)$ for all i, and then aggregates all the results into a single tag tag^* using Agg. (Since there is only one id here, we ignore it.)
  3. A sends tag^* to B.

- B receives tag^* and attempts to recover x. For $i = 1, \ldots, \ell$ it does:

  1. B sets $m_i = \langle i \rangle \| 0$ (this can be viewed as a guess that $x_i = 0$) and attempts to run $\mathsf{Vrfy}_k(\cdot, \mathsf{tag}^*)$. However, Vrfy expects to have as input some set of messages M from which it will read $t = O(\log \ell)$ messages (in addition to m_i). B simulates this by running $\mathsf{Vrfy}_k(\cdot, \mathsf{tag}^*)$ using all 2^t possibilities. (E.g., if $\mathsf{Vrfy}_k(\cdot, \mathsf{tag}^*)$ requests to read message m_j then B uses both $m_j = \langle j \rangle \| 0$ and $m_j = \langle j \rangle \| 1$.) We stress here that Vrfy may adaptively choose which messages to read, but this does not cause any problems.
  2. If Vrfy outputs 1 in any of the 2^t executions considered above, then B sets $x'_i = 0$; otherwise, it sets $x'_i = 1$.

- B outputs the value $x' = x'_1 \cdots x'_\ell$.

B's procedure is such that B sets $x'_i = 0$ iff there exists a subset of $O(\log \ell)$ messages such that Vrfy accepts $m_i = \langle i \rangle \| 0$ relative to this subset.

We claim that (for n sufficiently large) B recovers $x' = x$ with probability at least $3/4$, where the probability is taken over choice of k and A's input x. (We assume for simplicity that $\mathsf{Mac}, \mathsf{Agg}$, and Vrfy are deterministic, though one can check that this is not essential for our proof.) This implies that $|\mathsf{tag}^*| = \Omega(\ell)$.

Note first that, assuming perfect correctness of the aggregate MAC, the only possible error is when $x_i = 1$ but $x'_i = 0$ for some i. Assume that, for infinitely many values of n, an error of this type occurs with probability $p(n) \geq 1/4$ (where the probability is over choice of k and x). Then we construct an adversary $\mathcal{A}$ breaking the security of the aggregate MAC with probability at least $p/2^t\ell = O(1/\ell^2)$, a contradiction.

Adversary $\mathcal{A}$ works as follows. It chooses a random $x \in \{0,1\}^\ell$ and encodes x into a set M of ℓ messages exactly as A does. It then uses its Mac oracle to compute an aggregate MAC tag tag^* on the set M. $\mathcal{A}$ next chooses a random index i for which $x_i = 1$ and sets $\tilde{x}_i = 0$. It chooses the rest of the bits of $\tilde{x} \in \{0,1\}^\ell$ uniformly, and encodes $\tilde{x}$ into a set $\tilde{M}$ in the natural way. Finally, it outputs $\tilde{M}$ and tag^* along with the index i; in this context, $\mathcal{A}$ succeeds if verification of the ith message in $\tilde{M}$ (namely, the message $\langle i \rangle \| 0$) succeeds, given $\tilde{M}$ and tag^*. Note that $\mathcal{A}$ never submitted message $\langle i \rangle \| 0$ to its Mac oracle. It is easy to see that $\mathcal{A}$ runs in polynomial time, and that it succeeds with probability at least $p(n)/2^t$ as claimed. ■

In summary, it is not possible to do (much) better than our solution of the previous section when constant- or logarithmic-time verification is required. It is an interesting open question whether it is possible to do better than our construction when super-logarithmic (but sub-linear) verification time is desired.

Acknowledgments

Work of both authors was supported by US-Israel Binational Science Foundation grant #2004240. The work of the first author was additionally supported by NSF grant #0627306, and by the US Army Research Laboratory and the UK Ministry of Defence under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the US Government, the UK Ministry of 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.

References

[1] M. Bellare, O. Goldreich, and A. Mityagin. The power of verification queries in message authentication and authenticated encryption. Available at `http://eprint.iacr.org/2004/309`.

[2] M. Bellare, R. Guérin, and P. Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. In *Advances in Cryptology — Crypto '95*, volume 963 of *LNCS*, pages 15–28. Springer, 1995.

[3] R. Bhaskar, J. Herranz, and F. Laguillaumie. Aggregate designated verifier signatures and application to secure routing. *Intl. J. Security and Networks* 2(3/4): 192–201, 2007.

[4] A. Boldyreva, C. Gentry, A. O'Neill, and D. H. Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *14th ACM Conf. on Computer and Communications Security (CCS)*, pages 276–285. ACM Press, 2007.

[5] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in Cryptology — Eurocrypt 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, 2003.

[6] H. Chan, A. Perrig, and D. Song. Secure hierarchical in-network aggregation in sensor networks. In *ACM CCS '06: 13th ACM Conf. on Computer and Communications Security*, pages 278–287. ACM Press, 2006.

[7] O. Eikemeier, M. Fischlin, J.-F. Götzmann, A. Lehmann, D. Schröder, P. Schröder, and D. Wagner. History-free aggregate message authentication codes. In *7th Intl. Conf. on Security and Cryptography for Networks*, volume 6280 of *LNCS*, pages 309–328. Springer, 2010.

[8] C. Gentry and Z. Ramzan. Identity-based aggregate signatures. In *9th Intl. Conference on Theory and Practice of Public Key Cryptography(PKC 2006)*, volume 3958 of *LNCS*, pages 257–273. Springer, 2006.

[9] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[10] L. Hu and D. Evans. Secure aggregation for wireless networks. In *Workshop on Security and Assurance in Ad-Hoc Networks*, pages 384–394. IEEE, 2003.

[11] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. Sequential aggregate signatures and multisignatures without random oracles. In *Advances in Cryptology — Eurocrypt 2006*, volume 4004 of *LNCS*, pages 465–485. Springer, 2006.

[12] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. Sequential aggregate signatures from trapdoor permutations. In *Advances in Cryptology — Eurocrypt 2004*, volume 3027 of *LNCS*, pages 74–90. Springer, 2004.

[13] B. Przydatek, D. Song, and A. Perrig. SIA: Secure information aggregation in sensor networks. In *1st ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 255–265. ACM, 2003.

[14] H. Shacham. *New Paradigms in Signature Schemes*. PhD thesis, Stanford University, 2005.