Aggregate Message Authentication Codes

Jonathan Katz[1] and Andrew Y. Lindell[2]

[1] University of Maryland
jkatz@cs.umd.edu
[2] Aladdin Knowledge Systems and Bar-Ilan University
andrew.lindell@aladdin.com, 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. [5,16], 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 [12,11,6,13,4]. To the best of our knowledge, however, 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 shared-key) setting. Nevertheless, we suggest that aggregate MACs *can* be very 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 a highly "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 the sake of argument, let us say that messages are 16 bits long. (Replay attacks can be addressed by using a counter shared by the base station and all nodes in the network; 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, let us assume that 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 (approximately) $(160 + 16) \cdot (2t \log t) \approx 4.6 \times 10^7$ bits.

The above description assumes MACs used in the 'standard' manner, 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 $16(2t \log t) + 160t \approx 5.7 \times 10^6$ bits; this is 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 [14] or [9] 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 here 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. We remark that the existence of *efficient* aggregate MACs is somewhat surprising since algebraic (i.e., number-theoretic) properties of the underlying signature scheme are used to perform aggregation in the setting of aggregate signatures. 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 (basic construction – informally stated)**:** *If there exists a secure message authentication code, then there exists a secure aggregate message authentication code with complexity as outlined below.*

The complexity of our construction is as follows:

– *Aggregate MAC tag length:* equal to a single tag in 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 length of the tags to be aggregated
– *Verification of ℓ aggregated MACs:* equal to the time it takes to verify ℓ MACs in basic scheme

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. Nevertheless, the reason for this will become clear after seeing our construction in Section 3.

Lower bound. 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. (This is similar to the case of CBC encryption that requires the receiver to decrypt the entire message even if it only wants to read the last block.) 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 show 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 in the security parameter).

Related work. Subsequent to our work on this paper, we became aware of two other recent papers [7,3] 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.

2 Definitions

Our definitions are based on those given in [5,16] 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 *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 procedure 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* $\mathsf{Vrfy}_{k,\mathsf{id}}(m, \mathsf{Mac}_k(m)) = 1$. *(This is essentially the correctness condition for standard MACs.)*
- *Let* M^1, M^2 *be two sets of message/identifier pairs with*[2] $M^1 \cap M^2 = \emptyset$, *and let* $M = M^1 \cup M^2$. *If:*
  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 $\mathsf{Vrfy}_{(k_1,\mathsf{id}_1),\ldots,(k_n,\mathsf{id}_n)}(M, \mathsf{Agg}(M^1, M^2, \mathsf{tag}^1, \mathsf{tag}^2)) = 1$.

The second correctness condition states that the aggregation of MAC tags still enables correct verification.

The use of identifiers is merely a technical 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, in the second correctness condition, enforcing $M^1 \cap M^2 = \emptyset$ just means that aggregation is not applied if the same sender authenticated the same message twice.) Note that 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 $\mathsf{id}_i = i$ (it can be checked that this has no effect on our results).

An aggregate MAC would be used as follows. A receiver R who wants to receive authenticated messages from t senders begins by sharing uniformly random 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

[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.

tag tag^j computed by sender id_j on the message m_j, these two tags can be aggregated by computing the value $\mathsf{tag} \leftarrow \mathsf{Agg}(\{(m_i, \mathsf{id}_i)\}, \{(m_j, \mathsf{id}_j)\}, \mathsf{tag}^i, \mathsf{tag}^j)$. The receiver can then 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 [8]. 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 a non-uniform probabilistic polynomial-time adversary, and consider the following experiment involving $\mathcal{A}$ and parameterized by a security parameter n:*

- **Key generation:** *Keys $k_1, \ldots, k_t \in \{0,1\}^n$, for $t = \mathsf{poly}(n)$, 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** 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. *(We stress that all the pairs in M are required to be distinct.)*
- **Success determination:** *We say $\mathcal{A}$* 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 $\mathsf{Mac}(\mathsf{id}_{i^*}, m_{i^*})$.*

We say that the aggregate MAC scheme (Mac, Agg, Vrfy) *is* secure *if for all $t = \mathsf{poly}(n)$ and all non-uniform 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 the case of 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, 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 does not succeed in outputting 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 (and, in particular, MACs where messages have more than one valid tag for a given key), however, verification *cannot* be done this way. This means that certain randomized MACs (e.g., XOR-MAC [2]) cannot be utilized directly in the above construction, although we remark that any randomized MAC could be "derandomized" using a pseudorandom function. In any case, most commonly-used MACs are deterministic, and thus the restriction 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 a *deterministic* algorithm. (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^***:** *upon input* $k \in \{0,1\}^n$ *and* $m \in \{0,1\}^*$*, outputs* $\mathsf{Mac}_k(m)$*.*
– **Algorithm** Agg^***:** *upon input two sets* M^1, M^2 *of message/identifier pairs and two tags* $\mathsf{tag}^1, \mathsf{tag}^2$*, the algorithm outputs* $\mathsf{tag} = \mathsf{tag}^1 \oplus \mathsf{tag}^2$*.*
– **Algorithm** Vrfy^***:** *upon input a set of 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*, algorithm* Vrfy^* *computes* $\mathsf{tag}' = \bigoplus_{j=1}^{\ell} \mathsf{Mac}_{k_{i_j}}(m_j)$*, and outputs* 1 *if and only if* $\mathsf{tag}' = \mathsf{tag}$*. (We stress that the input to* Vrfy^* *is taken to be 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 and* Mac *is deterministic, 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 $i^* \leftarrow \{1, \ldots, t(n)\}$.
2. For $i = 1$ to $t(n)$:
   (a) If $i \neq i^*$, choose $k_i \leftarrow \{0,1\}^n$.
   (b) If $i = i^*$, do nothing (however, we *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 $\mathcal{F}$ aborts is exactly $1/t(n)$, 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 must be distinct.

This completes the proof. ■

Efficiency. Our construction for aggregate MACs is highly efficient. Consider the example of a mobile ad-hoc networks (MANET) as described in the introduction. If the nodes are arranged as a binary (or any other) 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 on, 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 a single message. To achieve authentication of a single message in *constant* time (i.e., independent of the number of aggregated tags ℓ), our approach yields a tag of length $\mathcal{O}(\ell \cdot T)$, where we take T to be the length of the tag in some underlying (standard) 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 equals concatenation). However, our approach yields a tradeoff where the product of the authentication time and tag length is $\mathcal{O}(\ell \cdot T)$. In the previous section, we achieved authentication in time ℓ with a tag of length T. At the other extreme, concatenating MAC tags gives authentication in constant time (i.e., requires verifying a single MAC) 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 logarithmic time (in ℓ, the number of aggregated MACs) must have a tag of length at least $\Omega(\ell)$.

We stress that in this section, we consider the running time as a function of the number ℓ of messages. Of course, it also takes time to compute and verify a single MAC tags. However, this is a fixed overhead for every value of the security parameter, and so what is really of interest is how many MAC tags need to be computed to verify a *single* message, when the number of aggregated MACs is ℓ.

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). Obviously, the only solutions of interest are those that are more efficient than verifying everything.

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 scheme, 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) $\ell' - 1$ other messages.

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 it is impossible to achieve a better result since the above is essentially optimal. Informally speaking, we show that if verification can be carried out in constant time (or even in time $\mathcal{O}(\log \ell)$), then the tag must be at least $\Omega(\ell)$ bits long.

Before proceeding further, we observe this does not contradict the positive result we 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

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

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

Claim 1. *Any aggregate MAC scheme in which verification of a single message can be carried out in time $\mathcal{O}(\log \ell)$ (where ℓ denote the total number of messages authenticated by an aggregate tag) has tags whose length is $\Omega(\ell)$.*

Proof: We begin by providing intuition as to why the claim is true. Assume that there exists an aggregate MAC scheme where verification of a single message takes time $\log \ell$, and the tags are of length less than ℓ. The main observation is that if verification takes time $\log \ell$, then the Vrfy algorithm can only read at most $\log \ell$ of the messages whose MACs are aggregated. If each of these messages consists of one bit only, then it is possible to try all possible combinations of the $\log \ell$ bits to see which passes verification (this takes time $2^{\log \ell} = \ell$ which is feasible). A key point here, of course, is that only a correct combination should pass or this could be used to efficiently construct a forgery of the aggregate MAC scheme. This implies that it is possible to reconstruct $\log \ell$ of the (single-bit) messages *given only the MAC tag*. However, this holds for all subsets of $\log \ell$ bits and so *all* of the messages can be reconstructed in polynomial-time given only the MAC tag. But this means that it is possible to reconstruct any ℓ-bit message from a tag of length less than ℓ. Stated differently, it means that an arbitrary ℓ-bit message can be compressed, something that is known to be impossible! Our formal proof follows this intuition with some minor changes, the main one being that we show how to reconstruct one bit at a time rather than blocks of $\log \ell$ bits. Furthermore, we derive our contradiction through lower bounds for probabilistic communication complexity rather than through compression; this is easier because of the negligible probability of error that exists when working with any cryptographic primitive.

We will use the *public random string model* of communication complexity, where two parties share a common random string and the question is how many bits must they communicate in order to correctly compute a function f. Given a protocol Π, the *error* of this protocol is given by $\max_{x,y}\{\Pr[\Pi(x,y) = f(x,y)]\}$ where the probability is taken over the parties' common random string as well as any internal randomness they might use. We let $CC_\epsilon(f)$ denote the minimum number of bits need to compute f, where this minimum is taken over all possible protocols with error at most ϵ. It is known that there exist functions $f : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ for which $CC_\epsilon(f) = \Omega(\ell)$; the inner-product function $IP(x,y) = \sum_{i=1}^{\ell} x_i y_i \bmod 2$ is one example. See [10,15] for more on communication complexity.

Let $(\mathsf{Mac}^*, \mathsf{Agg}^*, \mathsf{Vrfy}^*)$ be an aggregate MAC scheme in which verification of any message can be carried out in time $\mathcal{O}(\log \ell)$. At the very least, this implies that given any set of messages $M = \{m_1, \ldots, m_\ell\}$ and a single identifier id (using a single identifier just simplifies the proof), verification of a single message m_i (with respect to id) can be carried out by examining only $w = \mathcal{O}(\log \ell)$ other messages in M. (This is due to the fact that it is not possible to read more than $\log \ell$ messages in $\log \ell$ time.) We show that such a scheme implies that the probabilistic communication complexity (in the public random string model) of *every* function $f : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ is essentially the length of a tag for ℓ messages. However, since there exist functions with communication complexity $\Omega(\ell)$ (see the discussion in the previous paragraph) it follows that the tag length must also be $\Omega(\ell)$.

We begin by describing a protocol for computing any function $f : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ with communication complexity that is equal to the tag length plus 1. We stress that this protocol is for the setting of communication complexity and not cryptography. Thus, the parties A and B are fully honest and the only question is how many bits must be sent (there is no requirement on privacy, etc.). Loosely speaking, the protocol we describe works by having A compute an aggregate MAC on her input and then send the tag to B. Party B then reconstructs A's input from the tag, as described in the intuitive discussion above. Finally, given A's full input, B computes the output and sends it to A.

Before formally describing the protocol, we show how to encode a single ℓ-bit input into an aggregate MAC over ℓ messages. Let $x = x_1 \cdots x_\ell$ be A's input. Then, A defines messages $m_1, \ldots, m_\ell$ by $m_i = \langle i \rangle \| x_i$, where $\langle i \rangle$ is the binary encoding of i. The set $\{m_1, \ldots, m_\ell\}$ is a valid encoding of x because it fully defines x (all bits of x are represented, and their positions in x are given by the encoding of j that is included in every m_j). We remark that since only one id is used here, we ignore it from here on. We now describe the protocol:

Protocol 1. (communication complexity protocol for any function f)

- **Inputs:** *A has $x \in \{0,1\}^\ell$ and B has $y \in \{0,1\}^\ell$.*
- **Public random string:** *both parties share a random string $k \in \{0,1\}^n$ for some sufficiently large n.*
- **The protocol:**
  1. *A's first step:*
     (a) *Party A encodes its input $x = x_1 \cdots x_\ell$ into a set of ℓ messages $M = \{m_1, \ldots, m_\ell\}$ where $m_i = \langle i \rangle \| x_i$ for every i.*
     (b) *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^* by using the algorithm Agg^*.*
     (c) *A sends tag^* to B.*
  2. *Upon receiving tag^* from A, party B works as follows for $i = 1, \ldots, \ell$:*
     (a) *B sets $m_i = \langle i \rangle \| 0$ (this can be viewed as a guess that $x_i = 0$) and attempts to run $\mathsf{Vrfy}^*_k(M, \mathsf{tag}^*)$. However, Vrfy^* expects to receive M and in general may read up to $\log \ell$ other messages in M.[3] Therefore, B proceeds as follows:*
     (b) *Let $\mathsf{Vrfy}^*_k((j_1, m_{j_1}), \ldots, (j_t, m_{j_t}), \mathsf{tag}^*)$ be the algorithm defined by Vrfy^* after it has read the t messages indexed by $j_1, \ldots, j_t$ with content $m_{j_1}, \ldots, m_{j_t}$; note that m_i is not included in this notation as we assume it is read first. (Essentially, this algorithm is defined by fixing the prefix of its execution until this point.)*
     (c) *For $t = 0, \ldots, \log \ell$, B works as follows:*
        i. *If $t = \ell$, then return the output bit of*
        $$\mathsf{Vrfy}^*_k((j_1, m_{j_1}), \ldots, (j_t, m_{j_t}), \mathsf{tag}^*)$$
        ii. *Else , invoke $\mathsf{Vrfy}^*_k((j_1, m_{j_1}), \ldots, (j_t, m_{j_t}), \mathsf{tag}^*)$ and let j_{t+1} be the next message read by $\mathsf{Vrfy}^*_k((j_1, m_{j_1}), \ldots, (j_t, m_{j_t}), \mathsf{tag}^*)$.*
        iii. *Recursively invoke $\mathsf{Vrfy}^*_k((j_1, m_{j_1}), \ldots, (j_t, m_{j_t}), (j_{t+1}, 0), \mathsf{tag}^*)$ and $\mathsf{Vrfy}^*_k((j_1, m_{j_1}), \ldots, (j_t, m_{j_t}), (j_{t+1}, 1), \mathsf{tag}^*)$, and return the logical OR of their outputs.*

     If the output of Vrfy^* from the above procedure equals 1, then B sets $x_i = 0$. Otherwise, it sets $x_i = 1$.
  3. *Given $x = x_1, \ldots, x_\ell$, B computes $f(x, y)$ and returns the result to A.*
  4. *Both parties output $f(x, y)$.*

Note that B's procedure is such that if *any* of the recursive threads returns 1 then B sets $x_i = 0$. However, if this occurs, then this means that there exists a subset of $\log \ell$ messages $m_{j_1}, \ldots, m_{j_{\log \ell}}$ such that Vrfy accepts $m_i = \langle i \rangle \| 0$ relative to this subset. On the other hand, if this does not occur, then Vrfy rejects for all such sets, in which case B sets $x_i = 1$.

[3] Without loss of generality we assume that Vrfy first reads m_i and then up to $\log \ell$ other messages.

It is clear that if B reconstructs x correctly then the protocol is correct. It therefore remains to show that B correctly reconstructs x except with negligible probability. (Actually, in the context of communication complexity it suffices to show that this holds except with some constant probability. However, we show something stronger.)

We separately analyze the case that $x_i = 0$ and $x_i = 1$. In the case of $x_i = 0$ we have that A generated tag^* with $x_i = 0$ and some setting of the other bits. Therefore, there must exist some subset of $\log \ell$ messages that results in Vrfy accepting (this subset is defined by A's real input x). Thus, when $x_i = 0$, party B *always* sets $x_i = 0$. (This follows from the correctness condition of MACs that states that if a tag is correctly constructed, then Vrfy will always output 1.)

The more challenging case is that of $x_i = 1$. Assume that there exists a message $x = x_1, \ldots, x_\ell$ and an i such that with probability p, party B's procedure on an aggregate tag tag^* computed from x is such that $x_i = 1$ but B sets $x'_i = 0$. We use this to construct an adversary $\mathcal{A}$ that breaks the MAC scheme $(\mathsf{Mac}^*, \mathsf{Agg}^*, \mathsf{Vrfy}^*)$ with probability p/ℓ.[4] Adversary $\mathcal{A}$ encodes x into a set M exactly as party A does. It then uses its Mac^* oracle to compute an aggregate MAC on the set of ℓ messages M; let tag^* be the result. Next, $\mathcal{A}$ chooses uniformly distributed bits $b_1, \ldots, b_\ell \in \{0, 1\}$ and constructs a new set M' where $m'_i = \langle i \rangle \| 0$ and for all $j \neq i$, $m'_j = \langle j \rangle \| b_j$. Finally, $\mathcal{A}$ outputs the set M' and the tag tag^*. (We note that $\mathcal{A}$ uses oracles whereas the party A used the public random string k. However, party A's procedure does not use k in any way except to compute Mac^* legitimately and so $\mathcal{A}$ can simulate this using its Mac^* oracle.)

We claim that $\mathcal{A}$ succeeds in breaking the MAC scheme with probability p/ℓ. This is due to the following facts:

1. The set M' is such that $\mathcal{A}$ never queried $\mathsf{Mac}(m'_i)$ (because $m_i = \langle i \rangle \| 1$ but $m'_i = \langle i \rangle \| 0$).
2. $\mathcal{A}$ did not send any $\mathsf{Corrupt}$ queries
3. With probability $2^{-\log \ell} = 1/\ell$ the random bits chosen by $\mathcal{A}$ that are read by Vrfy_k are equal to those that result in B's procedure erring. Therefore,

$$\mathrm{Prob}[\mathsf{Vrfy}^*_k(M', \mathsf{tag}^*)] = \frac{p}{\ell}.$$

We conclude that $\mathcal{A}$ succeeds in its attack with probability p/ℓ, implying that p must be negligible (by the assumption that the scheme is secure). This implies that Protocol 1 (probabilistically) computes f with communication complexity $|\mathsf{tag}^*| + 1$. Since there exist functions f for which the communication complexity is $\Omega(\ell)$, this therefore implies that $|\mathsf{tag}^*| = \Omega(\ell)$ as required.

We conclude by remarking that it is actually only required that the adversary $\mathcal{A}$ run in polynomial-time in order to reach a contradiction regarding the MAC.

[4] We are being slightly informal here. What we prove is that for a given n and pair (x, i), a non-uniform adversary will succeed in breaking the MAC scheme with probability that is polynomially related to the probability that B's procedure errs regarding the ith bit. This will then imply that for all sufficiently large n's, the probability $p = p(n)$ must be negligible, as required.

In contrast, B can run in time 2^ℓ and this makes no difference (the bounds in communication complexity hold irrespective of the computational complexity of the parties). The crucial point is that $\mathcal{A}$ does *not* run B and so its complexity does not depend on B. Thus B could just try all 2^ℓ strings to see if one results in the MAC being accepted. The probability of $\mathcal{A}$ generating a successful forgery remains the same because it is simply based on a random guess. ■

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. An interesting question remains as to whether it is possible to do better than the tradeoff achieved by our construction when ℓ' is asymptotically larger than $\log \ell$. It would also be interesting to close the remaining multiplicative factor of T (the tag length of the underlying MAC).

Acknowledgments

The work of the first author was 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. Bellare, M., Goldreich, O., Mityagin, A.: The Power of Verification Queries in Message Authentication and Authenticated Encryption, `http://eprint.iacr.org/2004/309`
2. Bellare, M., Guérin, R., Rogaway, P.: XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 15–28. Springer, Heidelberg (1995)
3. Bhaskar, R., Herranz, J., Laguillaumie, F.: Aggregate Designated Verifier Signatures and Application to Secure Routing. Intl. J. Security and Networks 2(3/4), 192–201 (2007)
4. Boldyreva, A., Gentry, C., O'Neill, A., Yum, D.H.: Ordered Multisignatures and Identity-Based Sequential Aggregate Signatures, with Applications to Secure Routing. In: ACM CCCS (2007)
5. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003)
6. Gentry, C., Ramzan, Z.: Identity-Based Aggregate Signatures. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 257–273. Springer, Heidelberg (2006)
7. Chan, H., Perrig, A., Song, D.: Secure Hierarchical In-Network Aggregation in Sensor Networks. In: ACM CCCS, pp. 278–287 (2006)

8. Goldwasser, S., Micali, S., Rivest, R.: A Digital Signature Scheme Secure against Adaptive Chosen-Message Attacks. SIAM J. Computing 17(2), 281–308 (1988)
9. Hu, L., Evans, D.: Secure Aggregation for Wireless Networks. In: Workshop on Security and Assurance in Ad-Hoc Networks, pp. 384–394 (2003)
10. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press, Cambridge (1996)
11. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential Aggregate Signatures and Multisignatures Without Random Oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 465–485. Springer, Heidelberg (2006)
12. Lysyanskaya, A., Micali, S., Reyzin, L., Shacham, H.: Sequential Aggregate Signatures from Trapdoor Permutations. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 74–90. Springer, Heidelberg (2004)
13. Mu, Y., Susilo, W., Zhu, H.: Compact Sequential Aggregate Signatures. In: 2007 ACM Symposium on Applied Computing (SAC), pp. 249–253 (2007)
14. Przydatek, B., Song, D., Perrig, A.: SIA: Secure Information Aggregation in Sensor Networks. In: SenSys 2003, pp. 255–265 (2003)
15. Raz, R.: Lecture Notes on Circuit Complexity and Communication Complexity. IAS Summer School, `http://www.wisdom.weizmann.ac.il/~ranraz/lecturenotes/index.html`
16. Shacham, H.: New Paradigms in Signature Schemes. PhD Thesis, Stanford University (2005)