Abstract: It is common to implement challenge-response entity authentication with a MAC function. In such an entity authentication scheme, aggregate MAC is effective when a server needs to authenticate many entities. Aggregate MAC aggregates multiple tags (responses to a challenge) generated by entities into one short aggregate tag so that the entities can be authenticated simultaneously regarding only the aggregate tag. Then, all associated entities are valid if the pair of a challenge and the aggregate tag is valid. However, a drawback of this approach is that invalid entities cannot be identified when they exist. To resolve the drawback, we propose group-testing aggregate entity authentication by incorporating group testing into entity authentication using aggregate MAC. We first formalize the security requirements and present a generic construction. Then, we reduce the security of the generic construction to that of aggregate MAC and group testing. We also enhance the generic construction to instantiate a secure scheme from a simple and practical but weaker aggregate MAC scheme. Finally, we show some results on performance evaluation.

Keywords: entity authentication; message authentication; aggregate MAC; group testing

1. Introduction
1.1. Background

A MAC function is one of the most basic symmetric-key primitives for cryptography. Its typical application is challenge-response entity authentication, which assumes that a server and an entity share a secret key. In this scheme, the server first sends a challenge to the entity. Next, the entity computes a tag for the challenge using the MAC function with the shared secret key and returns it to the server. Finally, the server computes the tag in the same way and verifies the received tag.

Entity authentication is often crucial in identifying invalid entities to secure network applications and services. Additionally, a server may need to authenticate many devices simultaneously in an IoT network. In scenarios where an edge device plays the role of an aggregator, as shown in Figure 1, aggregate MAC [1] is suitable for efficient communication between the server and the aggregator for entity authentication. Aggregate MAC allows users to aggregate multiple tags into a tag so that the aggregate tag is as short as each of the multiple tags. In the situation shown in Figure 1, if the aggregator aggregates tags from devices and sends the aggregate tag to the server, then the server can authenticate the devices based only on the aggregate tag. If the aggregate tag is valid, then the server knows that all devices are valid. On the other hand, if the aggregate tag is invalid, then the server only knows that one or more invalid devices are included, which cannot be identified. The problem is if the server can identify invalid devices without knowing individual tags. As far as we know, it has not been addressed for entity authentication.
1.2. Our Contribution

We observe that group testing [2] can be employed to solve the above problem. For group testing, each item is assumed to be positive or negative. Multiple items are assumed to be able to be inspected by a test whose result is positive only if one or more positive items are included. All positive items can be identified with fewer tests than individual tests if the number of positive items is relatively small [3].

We introduce and explore group-testing aggregate authentication. It is a protocol participated in by multiple entities, an aggregator, and a server. Each entity has its own secret key shared with the server. The aggregator broadcasts a challenge from the server to the entities and collects their responses. The server then identifies the invalid entities by verifying the responses with the help of the aggregator.

We first formalized the scheme and its security requirements. The security requirements are impersonation resistance, completeness, and soundness. Impersonation resistance represents the notion that adversaries cannot impersonate an entity without knowing its secret key. Completeness requires that a valid response must not be judged invalid. Soundness requires that an invalid response must not be judged valid.

Furthermore, we present a generic construction combining a group-testing scheme and an aggregate MAC scheme. For each test in group testing, it aggregates the tags of entities examined by the test and verifies the aggregate tag. The aggregate tag is valid (negative) if all the involved tags are valid. Thus, invalid entities can be identified with fewer tests than by examining them individually. We also show that the generic construction satisfies impersonation resistance if the underlying aggregate MAC scheme is unforgeable, completeness if the underlying group-testing scheme satisfies completeness, and soundness if the underlying aggregate MAC scheme satisfies soundness. Furthermore, considering that the simple and practical Katz-Lindell aggregate MAC scheme [1] does not satisfy soundness, we enhance the generic construction to instantiate group-testing aggregate entity authentication satisfying soundness by using aggregate MAC not satisfying soundness.

Finally, we evaluate the performance of the proposed construction instantiated with SHA-256 [4] and HMAC [5] by software implementation.

1.3. Related Work


Hirose and Shikata [13,14] applied group testing to aggregate MAC for identifying invalid messages from multiple senders. They used non-adaptive group testing for a generic construction. Sato and Shikata [15] presented a generic construction using adaptive group
testing. Anada and Kamibayashi [16] followed the discussion by Sato and Shikata [17] and discussed the quantum security of aggregate MAC combined with non-adaptive group-testing. Ogawa et al. [18] presented a scheme reducing the number of aggregate tags based on biorthogonal codes.

1.4. Organization

Section 2 defines notations and cryptographic primitives and describes group testing. Section 3 provides the syntax and security requirements of aggregate MAC and its concrete schemes. Section 4 formalizes group-testing aggregate entity authentication and presents its generic construction, combining group-testing and aggregate MAC. Section 5 discusses the security of the generic construction and presents its enhancement. Section 6 shows some results of the performance evaluation by software implementation. Section 7 gives a brief concluding remark.

This article is an extended and improved version of our conference paper [19]. We refine the formalization of security requirements, which are described in Section 4, based on the idea by Bellare and Rogaway [20]. Accordingly, we revise the theorems and proofs, which are given in Section 5. We also add the results on performance evaluation.

2. Preliminaries

2.1. Notation

\{0, 1\}^l is regarded as the set of all binary sequences of length \(l\). Let \(\{0, 1\}^* := \bigcup_{l \geq 0} \{0, 1\}^l\). For binary sequences \(x, y\), their concatenation is denoted by \(x \| y\).

Let \(\mathcal{S}\) be a set. For \(v := (v_1, \ldots, v_n) \in \{0, 1\}^n\) and \(s := (s_1, \ldots, s_n) \in \mathcal{S}^n\), let \(v \oplus s := (s_{j_1}, \ldots, s_{j_w})\), where \(1 \leq j_1 < \cdots < j_w \leq n\) and \(v_j = 1\) iff \(j \in \{j_1, \ldots, j_w\}\). For \(u, u' \in \{0, 1\}^n\), let \(u \lor u'\) be their component-wise disjunction. Let \(s \leftarrow \mathcal{S}\) represent that \(s\) is sampled uniformly at random from \(\mathcal{S}\).

2.2. MAC Function and Pseudorandom Function

Let \(f : K \times \mathcal{X} \to \mathcal{Y}\) be a keyed function with its key space \(K\). \(f(K, \cdot)\) is often denoted by \(f_K(\cdot)\).

\(f\) is called a secure MAC function or unforgeable if it is intractable to predict unknown outputs of \(f_K\), where \(K \leftarrow K\). An adversary \(A\) is given a tagging oracle \(T_K^f\) and a verification oracle \(V_K^f\) and is allowed to make queries adaptively to them. In response to a query \(X \in \mathcal{X}\), \(T_K^f\) returns \(f_K(X)\). In response to a query \((X, Y) \in \mathcal{X} \times \mathcal{Y}\), \(V_K^f\) returns 1 if \(f_K(X) = Y\) and 0 otherwise. \(A\) is not allowed to ask \((X, Y)\) to \(V_K^f\) after asking \(X\) to \(T_K^f\). \(A_{T_K^f, V_K^f}\) is successful iff \(V_K^f\) returns 1 in response to at least one query. The advantage of \(A\) against \(f\) is

\[
\text{Adv}_{f}^{\text{MAC}}(A) := \Pr[A_{T_K^f, V_K^f} \text{ is successful}].
\]

\(f\) is called a secure pseudorandom function (PRF) if it is intractable to distinguish \(f_K\) with \(K \leftarrow K\) from a uniform random function \(\rho : \mathcal{X} \to \mathcal{Y}\). An adversary \(A\) is given either \(f_K\) or \(\rho\) as an oracle and is allowed to make adaptive queries in \(\mathcal{X}\). \(A\) outputs 0 or 1. The advantage of \(A\) against \(f\) is

\[
\text{Adv}_{f}^{\text{PRF}}(A) := \left| \Pr[A_{f_K} = 1] - \Pr[A_{\rho} = 1] \right|,
\]

where \(A\) is regarded as a random variable which takes values in \(\{0, 1\}\). It is easy to see that a secure PRF is a secure MAC function, and that a secure MAC function is not necessarily a secure PRF.
2.3. Cryptographic Hash Function

A cryptographic hash function $H : \{0,1\}^* \rightarrow \{0,1\}^r$ is often simply called a hash function. Among its various security requirements, our work is concerned with the random oracle model and collision resistance.

The random oracle model [21] assumes that $H$ is an ideal function such that, for any $X \in \{0,1\}^*$, $H(X)$ is chosen uniformly at random from $\{0,1\}^r$. $H$ is called a random oracle. $H$ is said to satisfy collision resistance if it is intractable to find a pair of distinct inputs of $H$ mapped to the same output. The advantage of an adversary $A$ against $H$ is

$$\text{Adv}_H^{col}(A) := \Pr[(X, X') \leftarrow A(H) : X \neq X' \land H(X) = H(X')]$$

Notice that the above definition is not theoretically precise: $H$ should be sampled from a sufficiently large number of hash functions at random.

2.4. Group Testing

Suppose that there exists a set of items, each of which is either positive or negative. It is assumed that a test can inspect multiple items simultaneously and that the result is positive if one or more positive items exist among them. Then, it may be possible to identify positive items with fewer tests than by inspecting all the items individually.

A group-testing algorithm can be described as a sequence of sets of tests. Suppose that there are $n$ items. Then, each test can be denoted by a vector in $\{0,1\}^n$ such that the $j$-th element equals 1 iff the test examines the $j$-th item. Let $G_1, G_2, \ldots, G_n \subseteq \{0,1\}^n$ be a sequence of sets of tests, where $u$ is the number of its stages. The sets of tests are conducted in this order, and the order of the tests in each stage is arbitrary. A group-testing algorithm is called non-adaptive if all the tests are determined beforehand. Thus, it has only a single stage. A group-testing algorithm is called adaptive if the tests in the next stage are determined after the tests in the current stage.

Let $G := G_1 \cup G_2 \cup \cdots \cup G_n$. It is reasonable to assume that each test examines at least one item and that the whole set of tests examines all items. Namely, $0^n \not\in G$ and $\bigcup_{G \in G} G = 1^n$. The group-testing algorithm extracts candidates of positive items in the following way. For $1 \leq j \leq n$, let $\text{id}_j$ denote the $j$-th item. For $1 \leq i \leq u$, let $G_i := \{g_i,1, g_i,2, \ldots, g_i,|G_i|\}$. Let $O(g) := \{|\text{id}_j| 1 \leq j \leq n \text{ and } g_j = 1\}$, where $g := (g_1, g_2, \ldots, g_n) \in \{0,1\}^n$.

1. $J_0 := \{\text{id}_1, \text{id}_2, \ldots, \text{id}_n\}$.
2. For $1 \leq i \leq u$, do the followings: (a) $J_i := J_{i-1}$; (b) For $1 \leq l \leq |G_i|$, if the result of $g_{i,l}$ is negative, then $J_i := J_i \setminus O(g_{i,l})$.
3. Output $J_u$.

We call the group-testing algorithm complete if $J_u$ does not include any negative elements. We call it sound if $J_u$ includes all the positive elements. It is sound if the results of the tests are always correct. On the other hand, it may not be complete in general.

For non-adaptive group testing, let us see the matrix $G$ whose rows are the vectors in a set $G$ of tests, which is called a group-testing matrix. We call $G$ $d$-disjunct if the component-wise disjunction of any $d$ columns in $G$ does not equal the component-wise disjunction of itself and any other single column. Suppose that non-adaptive group testing is represented by a $d$-disjunct matrix. Then, it is complete if there are at most $d$ positive items. Specifically, all the positive items are identified.

Suppose that there are at most $d$ positive items. For non-adaptive group testing, it is known that there exists a complete algorithm with $O(d^2 \log n)$ tests [3,22–24]. In addition, a non-asymptotic lower bound was conjectured as $\min\{(d+1)^2, n\}$ [25] while it is true for $d \leq 5$, and actually derived as $\min\{(d+2)(d+1)/2, n\}$ [26] and $\min\{d^2(15 + \sqrt{33})/24, n\}$ [27]. For adaptive group testing, it is known that there exists a complete algorithm with $O(d \log(n/d))$ tests [3,28,29]. A tight lower bound is shown as $d \log(n/d) + o(d \log(n/d))$ [3].
3. Aggregate MAC

3.1. Syntax

A tuple of algorithms \( \text{AM} \) := (\( \text{KG}, \text{Tag}, \text{Agg}, \text{Ver} \)) formalizes an aggregate MAC scheme. It is associated with an ID space \( \mathcal{I} \), a key space \( \mathcal{K} \), a message space \( \mathcal{M} \), a tag space \( \mathcal{T} \), and an aggregate-tag space \( \mathcal{A} \).

- \( \text{KG} \) is a key-generation algorithm such that \( k \leftarrow \text{KG}(\kappa) \), where \( \kappa \) is a security parameter and \( k \in \mathcal{K} \). Each entity is assigned a secret key independently generated by \( \text{KG} \).
- \( \text{Tag} \) is a tagging algorithm such that \( t \leftarrow \text{Tag}(k, m) \), where \((k, m) \in \mathcal{K} \times \mathcal{M} \) and \( t \in \mathcal{T} \).
- \( \text{Agg} \) is an aggregate algorithm such that \( T \leftarrow \text{Agg}((id_1, m_1, t_1), \ldots, (id_p, m_p, t_p)) \), where \((id_j, m_j, t_j) \in \mathcal{I} \times \mathcal{M} \times \mathcal{T} \) for \( 1 \leq j \leq p \) and \( T \in \mathcal{A} \). \((id_j, m_j)\)'s are required to be distinct from each other. It is often the case that \( T \) depends only on \( t_1, t_2, \ldots, t_p \).
- \( \text{Ver} \) is a verification algorithm such that \( d \leftarrow \text{Ver}(((id_1, k_1), \ldots, (id_p, k_p)), ((id_1, m_1), \ldots, (id_p, m_p)), T) \), where \((id_j, k_j) \in \mathcal{I} \times \mathcal{K} \) and \((id_j, m_j) \in \mathcal{I} \times \mathcal{M} \) for \( 1 \leq j \leq p \), \( T \in \mathcal{T} \) if \( p = 1 \) and \( T \in \mathcal{A} \) otherwise, and \( d \in \{0, 1\} \). \((id_j, m_j)\)'s are required to be distinct from each other. With respect to \(((id_1, k_1), \ldots, (id_p, k_p))\), the pair \(((id_1, m_1), \ldots, (id_p, m_p))\) and \( T \) are valid if \( d = 1 \) and invalid otherwise.

\( \text{AM} \) satisfies correctness. For \((id_1, k_1), \ldots, (id_p, k_p)\) and \((id_1, m_1), \ldots, (id_p, m_p)\), let \( t_j \leftarrow \text{Tag}(k, m) \) for \( 1 \leq j \leq p \) and \( T \leftarrow \text{Agg}((id_1, m_1, t_1), \ldots, (id_p, m_p, t_p)) \). Then, it holds that \( \text{Ver}(((id_1, k_1), \ldots, (id_p, k_p)), ((id_1, m_1), \ldots, (id_p, m_p)), T) = 1 \). In particular, for \( p = 1 \), if \( t \leftarrow \text{Tag}(k, m) \), then \( \text{Ver}((id, k), (id, m), t) = 1 \).

3.2. Security Requirement

Unforgeability and soundness are formalized as security requirements for aggregate MAC. Soundness is required for applying group testing to aggregate MAC [14].

3.2.1. Unforgeability

We introduce a game \( \text{Adv}^{uf}_{\text{AM,A}} \) to define unforgeability, where \( \text{A} \) is an adversary allowed to make queries adaptively to the oracles \( \mathcal{T} \mathcal{G}, \mathcal{K} \mathcal{D}, \text{and} \mathcal{V} \mathcal{R} \).

- \( \mathcal{T} \mathcal{G} \) is called a tagging oracle. It returns \( t \leftarrow \text{Tag}(k, m) \) in response to a query \((id, m)\), where \( k \) is the key of the entity \( id \).
- \( \mathcal{K} \mathcal{D} \) is called a key-disclosure oracle. It accepts a query \( id \) and returns \( k \).
- \( \mathcal{V} \mathcal{R} \) is called a verification oracle. It accepts a query \(((id_1, m_1), \ldots, (id_p, m_p)), T\) and returns \( d \leftarrow \text{Ver}(((id_1, k_1), \ldots, (id_p, k_p)), ((id_1, m_1), \ldots, (id_p, m_p)), T) \).

For a query \(((id_1, m_1), \ldots, (id_p, m_p)), T\) made by \( \text{A} \) to \( \mathcal{V} \mathcal{R} \), we call \((id_j, m_j)\) a fresh pair if \( \text{A} \) does not ask it to \( \mathcal{T} \mathcal{G} \) and does not ask \( id_j \) to \( \mathcal{K} \mathcal{D} \) prior to the query. \( \mathcal{V} \mathcal{R} \) does not accept a query with no fresh pair. \( \text{Adv}^{uf}_{\text{AM,A}} \) outputs 1 iff \( \text{A} \) gets 1 from \( \mathcal{V} \mathcal{R} \) for at least one query. The advantage of \( \text{A} \) against \( \text{AM} \) for unforgeability is defined by

\[
\text{Adv}^{uf}_{\text{AM,A}}(\text{A}) := \text{Pr}[\text{Adv}^{uf}_{\text{AM,A}} = 1].
\]

It is informally stated that \( \text{AM} \) is unforgeable or satisfies unforgeability if, for any efficient \( \text{A} \), \( \text{Adv}^{uf}_{\text{AM,A}}(\text{A}) \) is negligible.

3.2.2. Soundness

To define soundness, we specify a game \( \text{Adv}^{sd}_{\text{AM,A}} \), where \( \text{A} \) is an adversary allowed to make queries adaptively to the aggregate-then-verify oracle \( \mathcal{A} \mathcal{V} \mathcal{R} \) in addition to the oracles \( \mathcal{T} \mathcal{G}, \mathcal{K} \mathcal{D}, \text{and} \mathcal{V} \mathcal{R} \). \( \mathcal{A} \mathcal{V} \mathcal{R} \) accepts a query \(((id_1, m_1, t_1), \ldots, (id_p, m_p, t_p))\) and computes

1. \( d_j \leftarrow \text{Ver}(((id_j, k_j), (id_j, m_j), t_j)) \) for \( 1 \leq j \leq p \).
2. \( T \leftarrow \text{Agg}((id_1, m_1, t_1), \ldots, (id_p, m_p, t_p)) \), and
3. \( D \leftarrow \text{Ver}(((id_1, k_1), \ldots, (id_p, k_p)), ((id_1, m_1), \ldots, (id_p, m_p)), T) \).
Then, it returns \( D \land (\overline{d_1} \lor \overline{d_2} \lor \ldots \lor \overline{d_p}) \). \( \sigma^\text{AM}_A \) outputs 1 iff \( A \) gets 1 from \( AVR \) for at least one query. The advantage of \( A \) against \( AM \) for soundness is defined by

\[
\text{Adv}^\text{uf}_{\text{AM}}(A) := \Pr[\sigma^\text{uf}_{\text{AM},A} = 1].
\]

It is informally stated that \( AM \) is sound or satisfies soundness if, for any efficient \( A \), \( \text{Adv}^\text{uf}_{\text{AM}}(A) \) is negligible.

3.3. Aggregate MAC Scheme by Katz and Lindell

Let \( F : \mathcal{K} \times \mathcal{M} \rightarrow \{0, 1\}^\tau \) be a MAC function. The Katz-Lindell aggregate MAC scheme [1] using \( F \) is specified as follows:

- Each entity \( id \in \mathcal{I} \) is given a secret key \( k_{id} \leftarrow \mathcal{K} \).
- The tagging algorithm returns a tag \( t \leftarrow F_k(m) \) in response to \((k, m) \in \mathcal{K} \times \mathcal{M}\).
- The aggregate algorithm returns an aggregate tag \( T \leftarrow \bigoplus_{1 \leq i \leq p} t_i \) in response to \((id_1, m_1, t_1), \ldots, (id_p, m_p, t_p) \in \mathcal{I} \times \mathcal{M} \times \mathcal{T}\).
- Taking \((id_1, k_1), \ldots, (id_p, k_p) \) and \(((id_1, m_1), \ldots, (id_p, m_p), T) \) as input, the verification algorithm outputs 1 iff \( \bigoplus_{1 \leq i \leq p} F_{k_i}(m_i) = T \).

Let \( AM_X \) denote the Katz-Lindell aggregate MAC scheme. \( AM_X \) is shown to be unforgeable for any efficient adversary asking the verification oracle a single query [1]. It is also shown to be unforgeable even for any efficient adversary asking the verification oracle multiple queries:

**Proposition 1** ([14]). Let \( A \) be any adversary against \( AM_X \) with \( \ell \) users. Suppose that \( A \) asks the tagging oracle \( q_v \) queries and the verification oracle \( q_r \) queries. Suppose that each verification query by \( A \) consists of at most \( p \) pairs of ID and message. Then, there exists some adversary \( \bar{A} \) satisfying

\[
\text{Adv}^\text{uf}_{\text{AM}_X}(A) \leq \ell q_v \cdot \text{Adv}^\text{mac}_{F}(\bar{A}).
\]

\( \bar{A} \) asks the tagging oracle at most \((q_v + p)\) queries and the verification oracle at most one query. \( \bar{A} \)'s running time is at most about that of \( \sigma^\text{uf}_{\text{AM}_X,A} \).

It is easy to see that \( AM_X \) is not sound. Let \( \bar{A} \) be an adversary working as follows. \( \bar{A} \) first asks \((id_1, m_1) \) and \((id_2, m_2) \) to the tagging oracle and gets \( t_1 = F_{k_1}(m_1) \) and \( t_2 = F_{k_2}(m_2) \). Then, \( \bar{A} \) gets 1 from the aggregate-then-verify oracle by asking \(((id_1, m_1, t_1), (id_2, m_2, t_2)) \) such that \((t_1, t_2) \neq (t_1, t_2) \) and \( t_1 \oplus t_2 = t_1 \oplus t_2 \).

3.4. Aggregate MAC Scheme Using Hashing

We refer to the aggregate MAC scheme using a hash function \( H : \{0, 1\}^* \rightarrow \{0, 1\}^\tau \) to aggregate tags [14] as \( AH \). \( AH \) is specified as follows:

- The key generation and tagging algorithms are identical to those of \( AM_X \).
- For \((id_1, m_1, t_1), \ldots, (id_p, m_p, t_p) \), the aggregate algorithm returns \( T \leftarrow H(t_1 \parallel \ldots \parallel t_p) \).

For the uniqueness of the aggregate tag \( T \), \((id_1, m_1, t_1), \ldots, (id_p, m_p, t_p) \) are assumed to be ordered in a lexicographic order.

- Taking \((id_1, k_1), \ldots, (id_p, k_p) \) and \(((id_1, m_1), \ldots, (id_p, m_p), T) \) as input, the verification algorithm outputs 1 if \( H(F_{k_1}(m_1) \parallel \ldots \parallel F_{k_p}(m_p)) = T \) and 0 otherwise.

\( AH \) is shown to be unforgeable if \( F \) is unforgeable and \( H \) is a random oracle [14]:

**Proposition 2.** Let \( A \) be any adversary against \( AH \) with \( \ell \) users. Suppose that \( A \) asks the random oracle \( H \) \( q_h \) queries, the tagging oracle \( q_v \) queries, and the verification oracle \( q_r \) queries. Suppose that each verification query by \( A \) consists of at most \( p \) pairs of ID and message. Then, there exists some adversary \( \bar{A} \) satisfying

\[
\text{Adv}^\text{uf}_{\text{AH}}(A) \leq \ell q_v \cdot \text{Adv}^\text{mac}_{F}(\bar{A}) + q_v / 2^\tau.
\]
4.1. Scheme

4. Group-Testing Aggregate Entity Authentication

Step 4: With the help of the aggregator, the server identifies the valid entities using 

\[
\text{Tag}(\cdot, t_j) \quad \text{for } 1 \leq l \leq \vert G_l \vert \text{ and sends } (T_{i,1}, T_{i,2}, \ldots, T_{i,|G_l|}) \text{ to the server.}
\]

Step 3: The aggregator sends \((id_1, id_2, \ldots, id_n)\) to the server.

Step 4: With the help of the aggregator, the server identifies the valid entities using GT, Agg, and Ver in the following way:

1. \(J_0 \leftarrow \{id_1, id_2, \ldots, id_n\}\).
2. Let \(u\) be the number of stages of GT. For \(1 \leq i \leq u\),
   \(\begin{cases} 
   (a) & \text{According to GT, both the server and the aggregator determine the set of tests } G_i := \{g_1, \ldots, g_{|G_l|}\}. \\
   (b) & \text{The aggregator computes } T_{i,l} \leftarrow \text{Agg}(g_{ij} \oplus ((id_1, c, l_1), \ldots, (id_n, c, l_n))) \text{ for } 1 \leq l \leq |G_l| \text{ and sends } (T_{i,1}, T_{i,2}, \ldots, T_{i,|G_l|}) \text{ to the server.} \\
   (c) & \text{The server first sets } J_i \leftarrow J_{i-1}. \text{ Then, for } 1 \leq l \leq |G_i|, \text{ it computes } \\
   & D_{i,l} \leftarrow \text{Ver}(g_{ij} \oplus ((id_1, k_1), \ldots, (id_n, k_n)), g_{ij} \oplus ((id_1, c), \ldots, (id_n, c)), T_{i,l}) \\
   & \text{and } J_i \leftarrow J_i \setminus O(g_{ij}) \text{ if } D_{i,l} = 1. \text{ Finally, it sends } (D_{i,1}, D_{i,2}, \ldots, D_{i,|G_i|}) \text{ to the aggregator.} 
   \end{cases} \)
3. Output \(J_u\).

The communication among the server, the aggregator, and the entities in EA[GT, AM] is depicted in Figure 2.

For the description above, Step 3 can be merged with the first move of 2(b) in Step 4 if the server knows the number of entities to be authenticated in advance. If GT is non-adaptive, then \(u = 1\), and both the server and the aggregator know all the tests in advance. In addition, the server does not have to send the results of the tests to the aggregator in Step 4(c). If GT is adaptive, then the results of \(G_1, \ldots, G_i\) determine \(G_{i+1}\). Since the server sends the results of the current tests to the aggregator, the aggregator can also determine the new set of tests.
4.2. Security Requirement

The security requirements of $\text{EA}[\text{GT, AM}]$ are impersonation resistance, completeness, and soundness.

4.2.1. Impersonation Resistance

We introduce a game $\mathcal{G}_{\text{EA}[\text{GT, AM}], A}^{\text{im}}$ to formalize impersonation resistance. In this game, the adversary $A$ is supplied with oracles $\{S^{(i)} | i \in \mathbb{N}\}$ working as the server. For the $i$-th run of $\text{EA}[\text{GT, AM}]$, $A$ triggers $S^{(i)}$, which starts the protocol by returning a challenge $c^{(i)}$ to $A$. $A$ is also supplied with oracles $\{P^{(i)}_j | i \in \mathbb{N} \text{ and } 1 \leq j \leq n\}$ working as entities.

The $i$-th run of $\text{EA}[\text{GT, AM}]$ proceeds with the communication between $S^{(i)}$ and $A$. Multiple runs may proceed concurrently in general. Each $P^{(i)}_j$ accepts two kinds of queries. For a tagging query $(\text{tag}, c)$, $P^{(i)}_j$ returns $\text{Tag}(k_j, c)$. For a corrupt query $\text{corrupt}$, it returns $k_j$. Once $A$ gets $c^{(i)}$, $A$ is allowed to ask it only to $P^{(i)}_j$. At the end of the run, $S^{(i)}$ outputs the set $\mathcal{J}^{(i)}$ of IDs of invalid entities. $S^{(i)}$ may abort the run if $A$ does not follow the protocol.

$\mathcal{G}_{\text{EA}[\text{GT, AM}], A}^{\text{im}}$ outputs 1 iff there exist some $i^*$ and $j^*$ such that $id_{j^*} \not\in \mathcal{J}^{(i^*)}$. $A$ does not ask $c^{(i)}$ to $P^{(i)}_j$, and $A$ does not ask $\text{corrupt}$ to $P^{(i)}_j$ for any $i$. The advantage of $A$ against $\text{EA}[\text{GT, AM}]$ for impersonation resistance is

$$\text{Adv}_{\text{EA}[\text{GT, AM}]}^{\text{im}}(A) := \Pr[\mathcal{G}_{\text{EA}[\text{GT, AM}], A}^{\text{im}} = 1].$$

4.2.2. Completeness and Soundness

Completeness and soundness are security requirements for the identifiability of (in)valid responses to a challenge. We introduce games $\mathcal{G}_{\text{EA}[\text{GT, AM}], A}^{\text{cmp}}$ and $\mathcal{G}_{\text{EA}[\text{GT, AM}], A}^{\text{snd}}$. In both games, the adversary $A$ is not allowed to corrupt the server and the aggregator, and the communication channel between them is authenticated. Notice that, if $A$ is allowed to tamper aggregate tags, then any valid response by an entity can be judged invalid by the server.

In both of the games, the adversary $A$ is supplied with oracles $\{SA^{(i)} | i \in \mathbb{N}\}$ playing the roles of the server and the aggregator. $A$ is also supplied with oracles $\{P^{(i)}_j | i \in \mathbb{N}, 1 \leq j \leq n\}$ working as entities, which are specified in Section 4.2.1. For the $i$-th run of $\text{EA}[\text{GT, AM}]$, $A$ triggers $SA^{(i)}$, which starts the protocol by returning a challenge $c^{(i)}$ to $A$. Once $A$ gets $c^{(i)}$, $A$ is allowed to ask it only to $P^{(i)}_j$. In response to $c^{(i)}$, $A$ returns $(id_1, t^{(i)}_1), (id_2, t^{(i)}_2), \ldots, (id_n, t^{(i)}_n)$ to $SA^{(i)}$. $SA^{(i)}$ runs the protocol step by step. Each step is triggered by $A$. $A$ can also see the messages communicated during the protocol. At the end of the run, $SA^{(i)}$ outputs the set $\mathcal{J}^{(i)}$ of IDs of invalid entities. Multiple runs may proceed concurrently in general.
There exists some entity that the challenge $c^{(r)}$ of the $i^*$-th run of $\text{EA}[\text{GT}, \text{AM}]$ collides with some previous challenge $c^{(r')}$ ($i' < i^*$) or $c$ in a previous tagging query.

2. There exists some $i^*$ and $j^*$ such that, for some test $g := (g_1, \ldots, g_n) \in \{0, 1\}^n$ with $g_{j^*} = 1$ during the $i^*$-th run of $\text{EA}[\text{GT}, \text{AM}]$, $\text{Ver}(g \square ((id_1, k_1), \ldots, (id_n, k_n)), g \square$
Theorem 3. \((id_1, c^{(i_1)}), \ldots, (id_n, c^{(i_n)}), T^*) = 1\), and A does not ask \((\text{tag}, c^{(r)})\) to \(P_{j_r}^{(r)}\) and corrupt to any \(P_{j_r}^{(i)}\).

For the first case, notice that A triggers at most \(q_r\) runs of EA[GT, AM] and makes at most \(q_\ell\) tagging queries. Thus, the probability of the first case is at most

\[
(q_r(q_r - 1)/2) / 2^{
u} + q_r q_t / 2^\nu \leq q_r(q_r + 2q_t)/2^{\nu+1}.
\]

For the second case, \(\hat{A}\) gets 1 from VR in response to the query \((g \Box ((id_1, c^{(i_1)}), \ldots, (id_n, c^{(i_n)}), T^*)), (id_{j_r}, c^{(r)})\) is a fresh pair. Thus, \(\mathcal{F}_{AM}^{\text{Adv}_A}\) outputs 1. \(\square\)

5.2. Completeness and Soundness

It is easy to see that the completeness of EA[GT, AM] is reduced to the completeness of GT since AM satisfies correctness:

Theorem 2. If GT satisfies completeness, then, for any adversary A against EA[GT, AM],

\[
\text{Adv}_{\text{EA}[\text{GT, AM}]}^{\text{comp}}(A) = 0.
\]

Proof. Since GT satisfies completeness, for any valid tag, there exists some test such that it examines the tag and all the other tags it examines are valid. Since AM satisfies correctness, any aggregate tag generated only from valid tags is judged valid. \(\square\)

The soundness of EA[GT, AM] is reduced to the soundness of AM:

Theorem 3. For any adversary A against EA[GT, AM] for soundness, triggering at most \(q_\ell\) runs of EA[GT, AM] and making at most \(q_\ell\) tagging queries and \(q_\ell\) corrupt queries, there exists some adversary \(\hat{A}\) satisfying

\[
\text{Adv}_{\text{EA}[\text{GT, AM}]}^{\text{Comp}}(A) \leq \text{Adv}_{\text{AM}}^{\text{Comp}}(\hat{A}).
\]

Proof. In \(\mathcal{F}_{\text{AM}}^{\text{Adv}}\), \(\hat{A}\) runs \(\mathcal{F}_{\text{EA}[\text{GT, AM}]}^{\text{Adv}}\) in the similar way described in the proof of Theorem 1. \(\hat{A}\) simulates \(SA^{(1)}, SA^{(2)}, \ldots, SA^{(q_\ell)}\) by making use of VR. Suppose that A returns \((id_1, t_1^{(i_1)}), (id_2, t_2^{(i_2)}), \ldots, (id_n, t_n^{(i_n)})\) to SA\(^{(i)}\) in response to the challenge \(c^{(i)}\). Then, for each test \(g := (g_1, \ldots, g_n) \in \{0, 1\}^n\) during the \(i\)-th run of EA[GT, AM], \(\hat{A}\) also makes a query \(g \Box ((id_1, c^{(i_1)}), t_1^{(i_1)}), \ldots, (id_n, c^{(i_n)}), t_n^{(i_n)})\) to \(\mathcal{V R}\).

Suppose that \(\mathcal{F}_{\text{EA}[\text{GT, AM}]}^{\text{Adv}}(A)\) outputs 1 in \(\mathcal{F}_{\text{EA}[\text{GT, AM}]}^{\text{Adv}}(A)\). Then, there exists some \(i^*\) and \(j^*\) such that \(\{id_j | t_j^{(i)} \neq \text{Tag}(k_j, c^{(i)})\} \setminus J^{(i)} \neq \emptyset\). Thus, during the \(i^*\)-th run of EA[GT, AM], there exists some test \(g^* := (g_1^*, \ldots, g_n^*)\) with \(g_1^* = 1\) such that \(\text{Ver}(g^* \Box ((id_1, k_1), \ldots, (id_n, k_n)), g^* \Box ((id_1, c^{(i)}), \ldots, (id_n, c^{(i)})), T^{*}) = 1\) and \(t_j^{(i)} \neq \text{Tag}(k_j, c^{(i)})\), where \(T^* := \text{Agg}(g^* \Box ((id_1, c^{(i)}), t_1^{(i)}), \ldots, (id_n, c^{(i)}), t_n^{(i)}))\). Thus, \(\hat{A}\) gets 1 from \(\mathcal{V R}\) in response to \(g^* \Box ((id_1, c^{(i)}), t_1^{(i)}), \ldots, (id_n, c^{(i)}), t_n^{(i)})\). \(\square\)

5.3. Enhancing the Generic Construction

From the results so far, we confirm that EA[GT, AMX] and EA[GT, AMH] satisfy impersonation resistance and satisfy completeness if GT satisfies completeness. On the other hand, EA[GT, AMX] does not satisfy soundness, while EA[GT, AMH] satisfies soundness.
We enhance the proposed scheme and present EEA[GT, AM], which achieves soundness even with AMX.

EEA[GT, AM] is equipped with a PRF $R : \mathcal{R} \times \mathcal{I} \times \{0,1\}^{v+r} \rightarrow \{0,1\}^r$, where $\mathcal{R}$ is its key space. A shared secret key $r \in \mathcal{R}$ is given to the server and the aggregator. Notice that, for soundness, the communication channel between the server and the aggregator is assumed to be authenticated. Thus, the assumption is not critical that the server and the aggregator share a secret key. EEA[GT, AM] is specified as follows:

Steps 1 to 3: Identical to those of EA[GT, AM].
Step 4: $t_j \leftarrow R_r(id_j, c||t_j)$ for $1 \leq j \leq n$.
Step 5: Identical to Step 4 of EA[GT, AM].

The sole difference between EEA[GT, AM] and EA[GT, AM] is that the former utilizes $R$ to randomize the tags from the entities. Thus, Theorems 1 and 2 hold for EEA[GT, AM] as well as for EA[GT, AM]. In addition, EEA[GT, AMX] satisfies soundness if $R$ is a secure PRF:

**Theorem 4.** Let $A$ be any adversary against EEA[GT, AMX] for soundness. Suppose that $A$ triggers at most $q_\tau$ runs of EEA[GT, AMX] and makes at most $q_\rho$ tagging queries. Suppose that the runs of EEA[GT, AMX] conduct at most $q_\nu$ tests in total and $R$ is called at most $q_\mu$ times in total. Then, there exists some adversary $\hat{A}$ such that

$$\text{Adv}_{\text{EEA}[GT, AMX]}^{\text{snd}}(A) \leq \text{Adv}_{\text{EEA}[GT, AM]}^{\text{prf}}(A) + q_\tau^2/2^{\nu+1} + q_\rho/2^r.$$

$A$ makes at most $q_\mu$ queries to its oracle, and its running time is at most about that of $\text{Adv}_{\text{EEA}[GT, AMX]}^{\text{snd}}(A)$.

**Proof.** Let EEA$^\rho[GT, AMX]$ be identical to EEA[GT, AMX] except that the former uses $\rho : \mathcal{I} \times \{0,1\}^{v+r} \rightarrow \{0,1\}^r$ chosen uniformly at random instead of $R_r$ with $r \in \mathcal{R}$. The adversary $A$ against $R$ is given access to either $R_r$ or $\rho$. $A$ runs $\text{Adv}_{\text{EEA}[GT, AMX]}^{\text{snd}}(A)$ or $\text{Adv}_{\text{EEA}[GT, AMX]}^{\text{prf}}(A)$ with the use of $R_r$ or $\rho$, respectively. $A$ outputs 1 iff $A$ is successful for soundness. Then,

$$\text{Adv}_{\text{EEA}[GT, AMX]}^{\text{snd}}(A) \leq \text{Adv}_{\text{EEA}[GT, AM]}^{\text{prf}}(A) + \text{Adv}_{R}^{\text{prf}}(A).$$

Since

$$\text{Adv}_{R}^{\text{prf}}(A) = |\Pr[A_{R_r} = 1] - \Pr[A_{\rho} = 1]|$$

$$= |\text{Adv}_{\text{EEA}[GT, AMX]}^{\text{snd}}(A) - \text{Adv}_{\text{EEA}[GT, AMX]}^{\text{snd}}(A)|.$$

For $\text{Adv}_{\text{EEA}[GT, AMX]}^{\text{prf}}(A)$, let $\text{Co1}$ be the event that there exists a collision among the challenges generated in the runs of EEA$^\rho[GT, AMX]$. Then,

$$\text{Adv}_{\text{EEA}[GT, AMX]}^{\text{snd}}(A) \leq \Pr[\text{Co1}] + \Pr[\text{Adv}_{\text{EEA}[GT, AMX]}^{\text{snd}}(A) = 1 | \overline{\text{Co1}}].$$

Since $A$ triggers at most $q_\tau$ runs of EEA$[GT, AMX]$,

$$\Pr[\text{Co1}] \leq (q_\tau(q_\mu - 1)/2)/2^{\nu+1} \leq q_\tau^2/2^{\nu+1}.$$

Finally, let us see that

$$\Pr[\text{Adv}_{\text{EEA}[GT, AMX]}^{\text{snd}}(A) = 1 | \overline{\text{Co1}}] \leq q_\rho/2^r.$$

Let $c^{(i)}$ be the challenge in the $i$-th run of EEA$^\rho[GT, AMX]$ and $t_{i,j} := F_k(c^{(i)})$. If $\text{Co1}$ does not occur, then $\rho(id_j, c^{(i)}||t_{i,j})$ is chosen uniformly at random. Thus, the probability that the result of a test involving $(id_j, t_{i,j})$ such that $t'_{i,j} \neq t_{i,j}$ happens to be valid is at most $1/2^r$. \qed
6. Performance Evaluation

We implemented the verification algorithms of group-testing aggregate entity authentication for \( \text{EA}[\text{GT}, \text{AM}_X] \), \( \text{EEA}[\text{GT}, \text{AM}_X] \), and \( \text{EA}[\text{GT}, \text{AM}_H] \). We used the MAC function HMAC-SHA-256 for tagging and SHA-256 to aggregate tags for \( \text{AM}_H \). For \( \text{GT} \), we adopted non-adaptive group testing and used \( d \)-disjunct matrices generated by the shifted transversal design (STD) \[30\], where \( d \) is the upper bound on the number of invalid entities.

We implemented the algorithms in Python 3.10.9 and utilized the modules \texttt{hmac} and \texttt{hashlib} for SHA-256 and HMAC-SHA-256. We evaluated the performance of our implementations on a MacBook Pro with Apple M1, 16 GB of memory, and macOS Ventura 13.3.1.

The results are summarized in Table 1. For the numbers of the entities 100, 1000, and 10,000, the sizes of the matrices are \( 66 \times 100 \), \( 666 \times 1000 \), and \( 6969 \times 10,000 \), respectively. They are 5-, 17-, and 68-disjunct matrices, respectively.

Each time presented in Table 1 is the smallest of ten measurements. The “Tagging” column shows the time required to generate all the tags for the entities. Thus, they almost equal the time to verify all the tags of the entities one by one. For the same number of entities, there is no significant difference in the times required for verification by \( \text{EA}[\text{GT}, \text{AM}_X] \), \( \text{EEA}[\text{GT}, \text{AM}_X] \), and \( \text{EA}[\text{GT}, \text{AM}_H] \). They depend on the numbers of 1’s in the group-testing matrices, which are 600, 18,000, and 690,000 for 100, 1000, and 10,000 entities, respectively.

Table 1. Runtime (milliseconds).

<table>
<thead>
<tr>
<th>Number of Entities</th>
<th>Tagging</th>
<th>( \text{EA}[\text{GT}, \text{AM}_X] )</th>
<th>Verification EEA[GT, AM_X]</th>
<th>( \text{EA}[\text{GT}, \text{AM}_H] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>( 1.37 \times 10^{-1} )</td>
<td>( 2.65 \times 10^{-1} )</td>
<td>( 4.28 \times 10^{-1} )</td>
<td>( 3.05 \times 10^{-1} )</td>
</tr>
<tr>
<td>1000</td>
<td>( 7.83 \times 10^{-1} )</td>
<td>3.61</td>
<td>4.18</td>
<td>3.66</td>
</tr>
<tr>
<td>10,000</td>
<td>7.31</td>
<td>( 8.88 \times 10^{1} )</td>
<td>( 1.03 \times 10^{2} )</td>
<td>( 1.24 \times 10^{2} )</td>
</tr>
</tbody>
</table>

Figure 3 presents more details on the runtime for verification of \( \text{EEA}[\text{GT}, \text{AM}_X] \) with 1000 entities and \( 2 \leq d \leq 26 \). Table 2 presents the number of rows and the number of 1’s in the group-testing matrices used for the experiments. If \( d \geq 27 \), then \( \text{EEA}[\text{GT}, \text{AM}_X] \) cannot reduce the amount of communication between the server and the aggregator.

In Figure 3, the orange dots represent the times. For reference, we also give the blue dots representing the values of \((\text{the number of 1’s in the group-testing matrix})/5000\). As shown in Table 2, for group-testing matrices based on STD, the number of rows increases with the value of \( d \). On the other hand, this is not necessarily the case for the number of 1’s.

![Figure 3. Runtime for verification of EEA[GT, AM_X] with 1000 entities.](image)
Table 2. The number of rows and the number of 1's of $d$-disjunct matrices used for the experiments on Figure 3.

<table>
<thead>
<tr>
<th>$d$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td># rows</td>
<td>49</td>
<td>77</td>
<td>99</td>
<td>121</td>
<td>169</td>
<td>255</td>
<td>289</td>
<td>361</td>
<td>407</td>
</tr>
<tr>
<td># 1's</td>
<td>7000</td>
<td>7000</td>
<td>9000</td>
<td>11,000</td>
<td>13,000</td>
<td>15,000</td>
<td>17,000</td>
<td>19,000</td>
<td>11,000</td>
</tr>
<tr>
<td>$d$</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td># rows</td>
<td>444</td>
<td>481</td>
<td>518</td>
<td>555</td>
<td>592</td>
<td>629</td>
<td>666</td>
<td>703</td>
<td>740</td>
</tr>
<tr>
<td># 1's</td>
<td>12,000</td>
<td>13,000</td>
<td>14,000</td>
<td>15,000</td>
<td>16,000</td>
<td>17,000</td>
<td>18,000</td>
<td>19,000</td>
<td>20,000</td>
</tr>
<tr>
<td>$d$</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td># rows</td>
<td>777</td>
<td>814</td>
<td>851</td>
<td>888</td>
<td>925</td>
<td>962</td>
<td>999</td>
<td></td>
<td></td>
</tr>
<tr>
<td># 1's</td>
<td>21,000</td>
<td>22,000</td>
<td>23,000</td>
<td>24,000</td>
<td>25,000</td>
<td>26,000</td>
<td>27,000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7. Concluding Remark

We have introduced and explored group-testing aggregate authentication. We have first formalized the scheme and security requirements. Then, we have presented a general construction utilizing a group-testing scheme and an aggregate MAC scheme. We have reduced the security properties of the generic construction and its enhancement to those of the underlying group testing and aggregate MAC. Finally, we have shown results on the performance evaluation of the proposed construction instantiated with SHA-256 and HMAC.

The proposed construction can easily be deployed due to its simplicity. In addition, any progress in group testing and aggregate MAC will benefit it. Future work is to improve the performance further. It is interesting to see if the idea of Minematsu and Kamiya [12] is effective for our proposed construction.

Author Contributions: Conceptualization, S.H. and J.S.; methodology, S.H. and J.S.; software, S.H.; validation, S.H. and J.S.; formal analysis, S.H. and J.S.; investigation, S.H. and J.S.; resources, S.H.; data curation, S.H. and J.S.; writing—original draft preparation, S.H.; writing—review and editing, S.H. and J.S.; visualization, S.H.; supervision, J.S.; project administration, J.S.; funding acquisition, J.S. All authors have read and agreed to the published version of the manuscript.

Funding: These research results were obtained from the commissioned research (No.03901) by National Institute of Information and Communications Technology (NICT), Japan.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing is not applicable to this article.

Acknowledgments: We would like to thank Kazuhiko Minematsu for providing us disjunct matrices generated by using the STD method.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MAC Message authentication code
IoT Internet of things
XOR Exclusive or
PRF Pseudorandom function
STD Shifted transversal design
References

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.