Generic Construction of Sequential Aggregate MACs from Any MACs

Shingo Sato[1], Shoichi Hirose[2], and Junji Shikata[1](✉)

[1] Graduate School of Environment and Information Sciences, Yokohama National University, Yokohama, Japan
sato-shingo-cz@ynu.jp, shikata@ynu.ac.jp

[2] Faculty of Engineering, University of Fukui, Fukui, Japan
hrs_shch@u-fukui.ac.jp

Abstract. The aggregate message authentication code (aggregate MAC) is a cryptographic primitive which can compress MAC tags on multiple messages into a short aggregate MAC tag. Furthermore, the sequential aggregate MAC can check not only the validity of multiple messages but also the (sequential) order of messages. In this paper, we introduce a new model of sequential aggregate MACs where an aggregation algorithm generates a sequential aggregate tag depending only on any multiple and independent MAC tags with no secret-key, and we formally define security in this model. We also propose a generic construction of sequential aggregate MACs starting from various MACs without changing the structure of the MACs. This property is useful to make the existing networks more efficient by combining the aggregation algorithm with various MAC schemes already existing in the networks.

Keywords: Message authentication · MAC · Aggregate MAC
Sequential aggregate MAC

1 Introduction

The message authentication code (MAC) is one of the most fundamental cryptographic primitives. Furthermore, Katz and Lindell [8] proposed the aggregate MAC that can compress multiple MAC tags on multiple messages generated by different signers into a single aggregate tag. The advantage of the aggregate MAC lies in that the size of an aggregate tag is much smaller than total sizes of MAC tags, and hence it will be useful in applications in mobile networks or IoT (Internet of Things) networks where many devices sending messages are connected. The model and security of aggregate MACs were introduced by Katz and Lindell [8], and they proposed the simple construction satisfying the security by using exclusive-or of MAC tags.

Furthermore, there is another line of research about compressing MAC tags, called the sequential aggregate MACs. In sequential aggregate MACs, we can check not only the validity of multiple messages (like the aggregate MACs) but also the (sequential) order of messages. This property is required in applications including networks of resource-constrained devices such as IoT networks and mobile ad-hoc networks (MANET). Eikemeier et al. [5] formally defined the model and security for sequential aggregate MACs. They also introduced history-freeness which is a property depending only on a local message of each sender and the prior aggregate tag (aggregate-so-far tag), and they proposed history-free sequential aggregate MAC schemes. Ma and Tsudik [9] gave a simple construction by using hash functions for sequential aggregate MACs with forward security, however, they did not give a formal security proof to show that their construction met the security. Hence, Hirose and Kuwakado [6] formally defined the forward security in sequential aggregate MACs, and proposed a construction satisfying the security property with a formal security proof. Tomita et al. [10] gave a model of sequential aggregate authentication codes in the information-theoretic security setting, and they proposed constructions along with their model. The model in [10] focuses the one-time information-theoretic security which is different from those of [5,6,9].

Our motivation is to make the existing networks using MACs more efficient than the present state of affairs, however, it is not realistic to replace the currently existing network protocols with other ones entirely in general. Instead, we consider to simply embed a new node for improvement of efficiency (by aggregating MAC-tags sequentially) into the existing network without changing input-formats or structures of the existing MACs in the networks. In this paper, we call such a node an *aggregate node* whose role is to sequentially compress any multiple MAC-tags into a short tag without managing secret keys. The prior work for sequential aggregate MACs [5,6,9] does not satisfy our targeted property, namely, the prior work needs a new system setting (e.g., changing composition of MACs or setting an aggregate algorithm with a secret-key) or needs to change input-formats of the underlying MAC schemes (e.g., additional information with the local message would be required as input of MACs).

In this paper, we introduce a new model of sequential aggregate MACs where an aggregation algorithm generates a sequential aggregate tag depending only on multiple and independent MAC tags without any secret-key, and we formally define security in this model in Sect. 3. Our model and security are quite different from those of previous works. In addition, we propose two generic constructions of sequential aggregate MACs, called SAMAC1 and SAMAC2, starting from any MAC schemes (e.g., HMAC [2,3] and CMAC [1]) in Sect. 4.1, and we formally prove that our constructions meet the security in Sect. 4.2. We also show an application of our sequential aggregate MACs in Sect. 5: we consider a case where a device transmits long data by data-partitioning in a wireless network. Furthermore, it is shown in Sect. 6 that we can transform our construction into history-free sequential aggregate MACs [5]. Hence, ours can also be used as the prior sequential aggregate MACs.

To clarify the (dis)advantage of our constructions, we compare ours (i.e., SAMAC1 and SAMAC2) and the existing ones (i.e., MT [9], EFG+ [5], and HK [6]) in terms of *universal applicability*, *security*, and *efficiency* in Tables 1, 2 and 3, where we do not compare TWS [10] with others, since the security of TWS is information-theoretic and quite different from others. In the following, we explain differences among them by using Tables 1, 2 and 3.

(i) **Universal Applicability.** We consider applicability of embedding an *aggregate node* into the existing MAC protocols without changing the input-formats or network connections of underlying MACs. We also consider whether an aggregate algorithm can be executed without secret keys. Table 1 summarizes information about this. In previous works [5,6], each sender has to use not only a local message but also an aggregate-so-far tag to generate an aggregate tag, which means that we need to change the input-formats or structure of the underlying MACs. In addition, the constructions in [5,6,9] require other primitives except for MACs, such as a collision-resistant hash function [9], a pseudorandom permutation [5], or a pseudorandom generator [6]. On the other hand, our two constructions, SAMAC1 and SAMAC2, need not to change the input-formats or network connections of underlying MACs, and can generate an aggregate tag from MAC-tags without a secret key. While SAMAC1 requires only a MAC as a primitive, SAMAC2 needs a cryptographic hash function in addition to a MAC.

(ii) **Security.** We summarize provable security in Table 2. We note that a security proof of MT is not given, while other ones have provable security. We also note that the security proof of SAMAC2 is given in the random oracle model (ROM) while the security of HK, EFG+, and SAMAC1 are proved in the standard model (i.e., without random oracles).

(iii) **Efficiency.** Table 3 shows efficiency for the constructions. The number of function-calls, denoted by #Func.-call, shows how many times the required primitives are invoked in generating an aggregate tag. The number of function-calls in our constructions is smaller than those of the existing ones, which indicates that communication among senders and an aggregation node in our constructions is more efficient. Parallel computation in Table 3 means whether we can compute an aggregate tag in parallel. Although MT, HK, and EFG+ need to transmit an aggregate tag in a sequential way from a sender to another sender due to the order of messages, ours can compute an aggregate tag in parallel since each sender can compute a MAC-tag in parallel and then an aggregation node aggregates them into an aggregate tag following the order of messages. Parallel computability leads to less time complexity and avoids delay of sending messages in a network. Time complexity in Table 3 means the number of operations required for computing an aggregate tag. Our constructions do not need to compute MAC-tags N times owing to parallel computation of MAC-tags, while we need $(N-1)$ matrix multiplications in SAMAC1. It is not easy to strictly compare time complexity of SAMAC1 with those of MT, HK, and EFG+, since quite different operations are used. Anyway, we can say that our second construction SAMAC2 is best in time complexity since its time complexity does not depend on N. All of the constructions have the same bit-length of aggregate tags. The reduction loss being small implies that the gap between the resulting constructions and the underlying MACs in the security proof is small. From this viewpoint, SAMAC1 and SAMAC2 are superior to HK and EFG+. In total, we see that SAMAC2 is best among all ones in terms of efficiency.

(iv) **Summary.** In order to make the existing networks using MACs more efficient with slight change, universal applicability shown in Table 1 is important in a real world. In addition, we require provable security for the constructions, and we desire more efficiency than the current situation of the network that is our goal in this paper. From the viewpoints, we consider SAMAC2 is superior to others, though the security proof is given in the random oracle model. It is interesting to consider SAMAC1 as well in the standard model, and it is also interesting in versatility since it can be transformed into a history-free sequential aggregate MAC in the model of [5] without changing the input-formats of the MAC or adding any other primitive except for the MAC.

Table 1. Universal Applicability: CRH means a collision-resistant hash function, PRP means a pseudorandom permutation, PRG means a pseudorandom generator, and HF is a cryptographic hash function. MAC's input means the input-format required for the underlying MAC, m is a message, e is the end-marker in a time period, and $\tilde{\tau}$ is a previous aggregate tag.

Construction	Keyless aggregation	Primitive	MAC's input
MT [9]	✓	MAC and CRH	m
HK [6]		MAC and PRG	$m\|e\|\tilde{\tau}$
EFG+ [5]		MAC and PRP	m
SAMAC1	✓	MAC	m
SAMAC2	✓	MAC and HF	m

Table 2. Security: UF means unforgeability. ROM means the random oracle model, and Standard Model means the model without any random oracles.

Construction	Security level	Provable security	Standard model
MT [9]	Forward UF		n/a
HK [6]	Forward UF	✓	✓
EFG+ [5]	UF	✓	✓
SAMAC1	UF	✓	✓
SAMAC2	UF	✓	ROM

Table 3. Efficiency: Let N be the number of senders, let q be the number of queries to the tagging oracle, and let L be the maximum number of ID/message pairs in submitted queries. #Func.-call means the number that primitives are invoked in generating an aggregate tag. For a primitive $P \in \{\mathsf{MAC}, \mathsf{CRH}, \mathsf{PRG}, \mathsf{PRP}, \mathsf{HF}\}$, T_P means time complexity for computing P, and T_{MUL} means time complexity for computing multiplication of two matrices. Agg.-tag size means bit-length of aggregate tags, and n is bit-length of the underlying MAC. Reduction loss means the ratio ϵ/ϵ', where ϵ and ϵ' are the success probabilities of adversaries' attacks against the corresponding construction and the underlying MAC, respectively.

Construction	#Func.-call		Parallel computation	Time complexity	Agg.-tag size	Reduction loss
	Primitive	#Call				
MT [9]	MAC CRH	N N		$N \cdot T_{\mathsf{MAC}} + (N-1)T_{\mathsf{CRH}}$	n	n/a
HK [6]	MAC PRG	N U		$N \cdot T_{\mathsf{MAC}} + U \cdot T_{\mathsf{PRG}}$	n	$O(Nq^2)$
EFG+ [5]	MAC PRP	N N		$N \cdot T_{\mathsf{MAC}} + N \cdot T_{\mathsf{PRP}}$	n	$O(Nq^2L)$
SAMAC1	MAC	N	✓	$T_{\mathsf{MAC}} + (N-1)T_{\mathsf{MUL}}$	n	$O(N(1-2^{-\frac{n}{4}})^{-N})$
SAMAC2	MAC HF	N 1	✓	$T_{\mathsf{MAC}} + T_{\mathsf{HF}}$	n	$O(N)$

2 Preliminaries

In this paper, we use the following notations. For a positive integer n, let $[n] := \{1, 2, \ldots, n\}$. If we write a *negligible* function ε in λ, it means a function $\varepsilon : \mathbb{N} \to [0, 1]$ where $\varepsilon(\lambda) < 1/g(\lambda)$ for any polynomial g and a sufficiently large λ. We describe $\{x_i\}_{i\in[n]} := \{x_1, x_2, \ldots, x_n\}$ as a set of values x_i for all $i \in [n]$, and $(x_i)_{i\in[n]} := (x_1, x_2, \ldots, x_n)$ as a sequence of values x_i for all $i \in [n]$. We denote a polynomial in n by poly(n). Probabilistic polynomial time is abbreviated as PPT.

We define a deterministic message authentication code (MAC) as follows: A MAC scheme consists of three polynomial-time algorithms (KGen, Tag, Vrfy).

- $k \leftarrow \mathrm{KGen}(1^\lambda)$: KGen is a randomized algorithm which, on input a security parameter λ, outputs a secret key $k \in \mathcal{K}$.
- $t \leftarrow \mathrm{Tag}(k, m)$: Tag is a deterministic algorithm which, on input a secret key k and a message $m \in \mathcal{M}$, outputs a tag $t \in \mathcal{T}$.
- $1/0 \leftarrow \mathrm{Vrfy}(k, m, t)$: Vrfy is a deterministic algorithm which, on input a secret key k, a message m, and a tag t, outputs 1 (acceptance) or 0 (rejection).

Let $\mathcal{K}$ be a key-space, let $\mathcal{M}$ be a message-space, and let $\mathcal{T}$ be a tag-space. It is required that, for all $k \leftarrow \mathrm{KGen}(1^\lambda)$ and all $m \in \mathcal{M}$, we have $1 \leftarrow \mathrm{Vrfy}(k, m, \mathrm{Tag}(k, m))$.

We next define security notions of unforgeability against chosen message attacks (UF-CMA) and pseudorandomness for the MACs as follows: Let MAC = (KGen, Tag, Vrfy) be a MAC scheme.

UF-CMA. MAC meets UF-CMA, if for any PPT adversary $\mathcal{A}$ against MAC, the advantage of $Adv^{uf\text{-}cma}_{MAC,\mathcal{A}}(\lambda) := \Pr[\mathcal{A} \text{ wins}]$ is negligible, where $[\mathcal{A}$ wins$]$ is an event that $\mathcal{A}$ wins, in the following game:

- **Setup:** A challenger generates $k \leftarrow \mathrm{KGen}(1^\lambda)$ and sets $\mathcal{L}_{Tag} = \emptyset$.
- **Tagging:** The tagging oracle $\mathsf{Tag}_k(\cdot)$ takes a query $m \in \mathcal{M}$, returns $t \leftarrow \mathrm{Tag}(k, m)$, and sets $\mathcal{L}_{Tag} \leftarrow \mathcal{L}_{Tag} \cup \{m\}$. The number of queries submitted by $\mathcal{A}$ is at most $Q = \mathrm{poly}(\lambda)$.
- **Output:** When $\mathcal{A}$ outputs a forgery (m^*, t^*), $\mathcal{A}$ wins if the following holds: $1 \leftarrow \mathrm{Vrfy}(k, m^*, t^*)$, and $m^* \neq m$ for any $m \in \mathcal{L}_{Tag}$.

Pseudorandomness. MAC meets pseudorandomness, if the following holds: $Adv^{pr}_{MAC,\mathcal{D}}(\lambda) := \left|\Pr[\mathcal{D}^{\mathsf{Tag}_K(\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{D}^{f(\cdot)}(1^\lambda) = 1]\right|$ is negligible. Here, $\mathcal{D}$ is a PPT algorithm which, on input an oracle either $\mathsf{Tag}_k(\cdot)$ or $f(\cdot)$, determines which oracle is given; $\mathsf{Tag}_k(\cdot)$ is the tagging oracle which, on input $m \in \mathcal{M}$, returns $t = \mathrm{Tag}(k, m)$; and $f(\cdot)$ is an oracle which, on input $m \in \mathcal{M}$, returns $f(m)$ for a random function $f : \mathcal{M} \to \mathcal{T}$.

3 Sequential Aggregate MACs: Our Model and Security

We introduce a new model of sequential aggregate MACs where an aggregation algorithm generates a sequential aggregate tag depending only on multiple and independent MAC tags without any secret-key, and we formally define security in this model.

Let MAC=(KGen, Tag, Vrfy) be a MAC scheme. Then, a sequential aggregate MAC scheme consists of a tuple of five polynomial-time algorithms (KGen, Tag, Vrfy, SeqAgg, SAVrfy) as follows, where N is the number of senders, $\mathcal{ID}$ is an ID-space, $\mathcal{K}$ is a key-space, $\mathcal{M}$ is a message-space, $\mathcal{T}$ is a tag-space, and $\mathcal{T}_{agg}$ is an aggregate tag-space. Let $\mathcal{S} := \{(id_{\ell_1}, id_{\ell_2}, \ldots, id_{\ell_{\widehat{N}}}) \mid \widehat{N} \leq N \wedge id_i \neq id_j \text{ if } i \neq j\}$, which means the set of all different sequences of IDs with length at most N:

- $k_{id} \leftarrow \mathrm{KGen}(1^\lambda, id)$: KGen is a randomized algorithm which, on input a security parameter λ and an ID $id \in \mathcal{ID}$, outputs a secret key $k_{id} \in \mathcal{K}$. Note that this is the same as KGen of the underlying MAC except for adding id.
- $t \leftarrow \mathrm{Tag}(k_{id}, m)$: Tag is a deterministic algorithm which, on input a secret key k_{id} and a message m, outputs a tag $t \in \mathcal{T}$. This is the same as Tag of the MAC.
- $1/0 \leftarrow \mathrm{Vrfy}(k_{id}, m, t)$: Vrfy is a deterministic algorithm which, on input a secret key k_{id}, a message $m \in \mathcal{M}$, and a tag t, outputs 1 (acceptance) or 0 (rejection). This is the same as Vrfy of the MAC.
- $\tau \leftarrow \mathrm{SeqAgg}(T)$: SeqAgg is a deterministic algorithm which, on input a sequence of tags $T = ((id_{\ell_i}, t_i))_{i \in [\widehat{N}]}$ such that $(id_{\ell_1}, \ldots, id_{\ell_{\widehat{N}}}) \in \mathcal{S}$, outputs an aggregate tag $\tau \in \mathcal{T}_{agg}$.

- $1/0 \leftarrow$SAVrfy(K, M, τ): SAVrfy is a deterministic algorithm which, on input a set of key/id pairs $K = \{(k_{id_i}, id_i)\}_{i\in[N]}$, a sequence of message/id pairs $((m_i, id_{\ell_i}))_{i\in[\widehat{N}]}$ for any $(id_{\ell_1}, \ldots, id_{\ell_{\widehat{N}}}) \in \mathcal{S}$, and an aggregate tag τ, outputs 1 (acceptance) or 0 (rejection).

We require that the following condition (i.e., correctness) holds:

- For all $id \in \mathcal{ID}$, all $k_{id} \leftarrow$ KGen$(1^\lambda, id)$ and all $m \in \mathcal{M}$, we have $1 \leftarrow$ Vrfy$(k_{id}, m, \text{Tag}(k_{id}, m))$.
- For all $id \in \mathcal{ID}$, all $k_{id} \leftarrow$ KGen$(1^\lambda, id)$ and all $m \in \mathcal{M}$, for any $K = \{(k_{id_i}, id_i)\}_{i\in[N]}$ and any $M = ((m_i, id_{\ell_i}))_{i\in[\widehat{N}]}$ such that $(id_{\ell_1}, \ldots, id_{\ell_{\widehat{N}}}) \in \mathcal{S}$, we have $1 \leftarrow$ SAVrfy(K, M, τ), where $T = ((id_{\ell_i}, \text{Tag}(k_{id_{\ell_i}}, m_i)))_{i\in[\widehat{N}]}$ and $\tau = \text{SeqAgg}(T)$.

We define the following relation for sequences of message/ID pairs in order to define security of sequential aggregate MACs in our model.

Definition 1. *For two sequences of message/ID pairs* $M^{(1)} = ((m_i^{(1)}, id_i^{(1)}))_{i\in[N^{(1)}]}$ *and* $M^{(2)} = ((m_i^{(2)}, id_i^{(2)}))_{i\in[N^{(2)}]}$, *we define* $(M^{(1)})_{i_1,i_2} \equiv (M^{(2)})_{i'_1,i'_2}$ *for* i_1, i_2, i'_1, i'_2 *such that* $i_1 < i_2 \le N^{(1)}$ *and* $i'_1 < i'_2 \le N^{(2)}$, *if the following holds:*

$$((m_{i_1}^{(1)}, id_{i_1}^{(1)}), \cdots, (m_{i_2}^{(1)}, id_{i_2}^{(1)})) = ((m_{i'_1}^{(2)}, id_{i'_1}^{(2)}), \cdots, (m_{i'_2}^{(2)}, id_{i'_2}^{(2)})).$$

If not, we denote $(M_1)_{i_1,i_2} \not\equiv (M_2)_{i'_1,i'_2}$.

We next define a security notion of *C-aggregate unforgeability against chosen message attacks* (C-aggUF-CMA) in our model.

Definition 2 (C-aggUF-CMA). *A sequential aggregate MAC scheme* SAMAC = (KGen, Tag, Vrfy, SeqAgg, SAVrfy) *meets C-aggUF-CMA, if for any PPT adversary $\mathcal{A}$ against* SAMAC, *the advantage* $Adv^{\text{agg-uf}}_{\text{SAMAC},\mathcal{A}}(\lambda) :=$ Pr[$\mathcal{A}$ *wins*] *of $\mathcal{A}$ is negligible, where* [$\mathcal{A}$ *wins*] *is an event that $\mathcal{A}$ wins, in the following game:*

Setup: *A challenger generates a set of secret-key/ID pairs* $K = \{(k_{id_i}, id_i)\}_{i\in[N]}$ *by using the KGen algorithm. Then, it sets lists* $\mathcal{L}_{Cor} = \emptyset$ *and* $\mathcal{L}_{SA} = \emptyset$.

Corrupt: *The corrupt oracle* Corrupt$(\cdot)$ *takes an ID id* $\in \mathcal{ID}$ *as input and returns the secret key* k_{id} *and sets* $\mathcal{L}_{Cor} \leftarrow \mathcal{L}_{Cor} \cup \{id\}$, *where* $\mathcal{L}_{Cor}$ *means a list of IDs whose corresponding secret keys are known by an adversary. The number of queries submitted by $\mathcal{A}$ is at most C.*

Tagging: *The sequential aggregate tagging oracle* $\mathsf{SATag}_K(\cdot)$ *takes a sequence of message/ID pairs* $M = ((m_i, id_{\ell_i}))_{i\in[\widehat{N}]}$ *such that* $(id_{\ell_1}, \ldots, id_{\ell_{\widehat{N}}}) \in \mathcal{S}$ *and the number of not corrupted tags is at least $N - C$, where without loss of generality, we assume that* $id_{\ell_1} \notin \mathcal{L}_{Cor}$ *and* $id_{\ell_{\widehat{N}}} \notin \mathcal{L}_{Cor}$. *Then, it does the following:*

1. Compute $t_i \leftarrow \mathrm{Tag}(k_{id_{\ell_i}}, m_i)$ *for all* $i \in [\widehat{N}]$*,*
2. Output $\tau \leftarrow \mathrm{SeqAgg}(((id_{\ell_i}, t_i))_{i\in[\widehat{N}]})$*,*
3. Set $\mathcal{L}_{SA} \leftarrow \mathcal{L}_{SA} \cup \{M\}$*.*

The number of queries which $\mathcal{A}$ *submits is at most* $Q = \mathrm{poly}(\lambda)$*.* $\mathcal{A}$ *is not allowed to access* Corrupt$(\cdot)$ *after accessing* $\mathrm{SATag}_K(\cdot)$*. In addition,* $\mathcal{A}$ *is not allowed to query* M *such that for any* $M' \in \mathcal{L}_{SA}$*, it holds that* $(M)_{1,1+\ell} \equiv (M')_{1,1+\ell}$ *or* $(M)_{\widehat{N}-\ell,\widehat{N}} \equiv (M')_{\widehat{N}'-\ell,\widehat{N}'}$*, where* $\ell := \min(\widehat{N}-1, \widehat{N}'-1)$ *and* $\widehat{N}'$ *is the number of message/ID pairs in* M'*.*

Output: *When* $\mathcal{A}$ *outputs* $M^* = ((m_i^*, id_{\ell_i^*}))_{i\in[\widetilde{N}]}$ *and* τ^**,* $\mathcal{A}$ *wins if the following holds:*

- $1 \leftarrow \mathrm{SAVrfy}(K, M^*, \tau^*)$*,*
- $(id_{\ell_1^*}, \ldots, id_{\ell_{\widetilde{N}}^*}) \in \mathcal{S}$ *such that* $id_{\ell_1^*} \notin \mathcal{L}_{Cor}$ *and* $id_{\ell_{\widetilde{N}}^*} \notin \mathcal{L}_{Cor}$*, and the number of not corrupted IDs is at least* $N - C$*,*
- $M^* \notin \mathcal{L}_{SA}$*, and* M^* *is not any concatenation of queries in* $\mathcal{L}_{SA}$ *and a tag generated by a secret key of corrupted entities in* $\mathcal{L}_{Cor}$*.*

In principle, it is impossible in our model to guarantee the unforgeability against an adversary who can observe each MAC-tag before the aggregation. This reason is that, if the adversary obtains a sequence $\{(m_i, \mathsf{Tag}(k_{id_i}, m_i))\}_{i\in[\widehat{N}]}$ by accessing the tagging oracle, he can generate an aggregate tag for any sequential messages $(m_{\ell_i})_{i\in[N]}$ because SeqAgg algorithm is keyless. Thus, we consider the attacking model where an adversary makes a forgery by only accessing the sequential aggregate tagging oracle.

We next show a condition for a SeqAgg algorithm to achieve C-aggUF-CMA. For simplicity, we view a SeqAgg algorithm as a function $F : \mathcal{T}^N \to \mathcal{T}_{agg}$, where $\mathcal{T}$ is a MAC tag-space and $\mathcal{T}_{agg}$ is an aggregate tag-space. Then, the following proposition states that, for given $y \in \mathcal{T}_{agg}$, the equation $F(x) = y$ should not be correctly solved in polynomial time for achieving C-aggUF-CMA.

Proposition 1. *Let* SAMAC = (KGen, Tag, Vrfy, SeqAgg, SAVrfy) *be a sequential aggregate MAC scheme, and we identify SeqAgg with* $F : \mathcal{T}^N \to \mathcal{T}_{agg}$*. If we can compute a sequence of tags* $(t_i)_{i\in[N]}$ *from* $F((t_i)_{i\in[N]})$ *in polynomial time, then SAMAC does not meet C-aggUF-CMA.*

Proof. Let $\mathcal{A}$ be a PPT adversary against SAMAC. If $\mathcal{A}$ gets an aggregate tag τ on a sequence of messages $(m_i)_{i\in[N]}$ by accessing the sequential aggregate tagging oracle, $\mathcal{A}$ can compute each MAC-tag t_i on m_i for all $i \in [N]$. For a message sequence $(m_{\ell_i})_{i\in[N]}$ different from $(m_i)_{i\in[N]}$, $\mathcal{A}$ can make a forgery $\tau^* = F((t_{\ell_i})_{i\in[N]})$ on $(m_{\ell_i})_{i\in[N]}$. □

4 Construction of Sequential Aggregate MACs

4.1 Our Construction

We propose a generic construction of sequential aggregate MACs (SAMACs) in our model such that our SAMAC consists of any MAC scheme MAC =

(MAC.KGen, MAC.Tag, MAC.Vrfy) and a sequential aggregation algorithm-pair (SA.SeqAgg, SA.SAVrfy). This is well explained by the following construction of sequential aggregate MACs, GSAMAC = (KGen, Tag, Vrfy, SeqAgg, SAVrfy):

- $k_{id} \leftarrow \mathrm{KGen}(1^\lambda, id)$: Generate $k_{id} \leftarrow \mathrm{MAC.KGen}(1^\lambda)$ and output k_{id}.
- $t \leftarrow \mathrm{Tag}(k_{id}, m)$: Output a MAC-tag $t \leftarrow \mathrm{MAC.Tag}(k_{id}, m)$.
- $1/0 \leftarrow \mathrm{Vrfy}(k_{id}, m, t)$: Output $b \leftarrow \mathrm{MAC.Vrfy}(k_{id}, m, t) \in \{0, 1\}$.
- $\tau \leftarrow \mathrm{SeqAgg}(T)$: Take a sequence of MAC tags $T = ((id_{\ell_i}, t_i))_{i \in [\widehat{N}]}$ as input and output $\tau \leftarrow \mathrm{SA.SeqAgg}(T)$.
- $1/0 \leftarrow \mathrm{SAVrfy}(K, M, \tau)$: Take $K = \{(k_{id_i}, id_i)\}_{i \in [N]}$, $M = ((m_i, id_{\ell_i}))_{i \in [\widehat{N}]}$, and τ as input, and output a bit $b \leftarrow \mathrm{SA.SAVrfy}(K, M, \tau)$.

Therefore, it is enough to construct only a sequential aggregation algorithm-pair (SA.SeqAgg, SA.SAVrfy), and we propose two constructions called SA1 and SA2 for it. Consequently, we will obtain two constructions of SAMACs called SAMACi ($i = 1, 2$) starting from any MAC scheme and the aggregate algorithm-pair SAi.

We construct two aggregation algorithm-pairs SA1 and SA2. First, we describe the basic processes of SA1 informally as follows.

- SeqAgg algorithm:
  1. Each MAC-tag t_i is transformed into a matrix T_i,
  2. Output the product of these matrices $T_1 T_2 \cdots T_{\widehat{N}}$ as an aggregate tag τ.
- SAVrfy algorithm:
  1. Compute an aggregate tag τ' from $K = \{(k_{id_i}, id_i)\}_{i \in [N]}$ and $M = ((m_i, id_{\ell_i}))_{i \in [\widehat{N}]}$ following the SeqAgg algorithm,
  2. Output 1 (accept) if $\tau' = \tau$, or output 0 (reject) otherwise.

Our idea is based on non-commutativity of matrix multiplications. And, the order of messages is regarded as invalid, if the order of MAC-tags' matrices are changed. From this, we can construct sequential aggregate MAC schemes from any MAC schemes by transforming each MAC-tag into a matrix, and can give a security proof in the standard model. Also, we provide a simple construction SA2 by using hash functions, and give a security proof in the random oracle model. Furthermore, based on SA1, we can construct a history-free sequential aggregate MAC scheme from any MACs in Sect. 6. Although the existing construction [5] uses not only MACs but also pseudorandom permutations for constructing history-free sequential aggregate MACs, our construction requires only MACs.

Besides, it should be noted that we cannot achieve the security under consideration even if we slightly change inputs of the underlying MACs as follows: Suppose that, for each $i \in [N]$, the i-th sender computes $t_i \leftarrow \mathrm{Tag}(k_{id_i}, i \parallel m_i)$ and the resulting aggregate tag is $\tau = t_1 \oplus \cdots \oplus t_N$. However, it is easy to generate a valid forgery in the case where some IDs are corrupted. Actually, for an aggregate tag $\tau = t_1 \oplus \cdots \oplus t_N$ on $(m_i)_{i \in [N]}$, an adversary can compute $t'_i \leftarrow \mathrm{Tag}(k_{id_i}, i \parallel m'_i)$ with a corrupted ID's (i.e., id_i) secret key k_{id_i} and can generate a forgery $\tau' = \tau \oplus t_i \oplus t'_i$ without accessing the tagging oracle. Furthermore, even if an adversary does not corrupt any secret keys, he can break aggUF-CMA by submitting queries to the tagging oracle and receiving the following aggregate-tags:

$$\begin{aligned}\tau_1 &= \mathrm{Tag}(k_{id_1}, 1 \parallel m_1) \oplus \mathrm{Tag}(k_{id_2}, 2 \parallel m_2) \oplus \mathrm{Tag}(k_{id_3}, 3 \parallel m_3) \oplus \cdots,\\ \tau_2 &= \mathrm{Tag}(k_{id_1}, 1 \parallel m_1') \oplus \mathrm{Tag}(k_{id_2}, 2 \parallel m_2) \oplus \mathrm{Tag}(k_{id_3}, 3 \parallel m_3) \oplus \cdots,\\ \tau_3 &= \mathrm{Tag}(k_{id_1}, 1 \parallel m_1) \oplus \mathrm{Tag}(k_{id_2}, 2 \parallel m_2') \oplus \mathrm{Tag}(k_{id_3}, 3 \parallel m_3) \oplus \cdots.\end{aligned}$$

Then, he computes $\tau_1 \oplus \tau_2 \oplus \tau_3 = \mathrm{Tag}(k_{id_1}, 1 \parallel m_1') \oplus \mathrm{Tag}(k_{id_2}, 2 \parallel m_2') \oplus \mathrm{Tag}(k_{id_3}, 3 \parallel m_3) \oplus \cdots$, which is a valid forgery since the sequence $((m_1', id_1), (m_2', id_2), (m_3, id_3), \ldots)$ has never been queried. Therefore, this construction does not meet the security of sequential aggregate MACs in our model.

Construction 1. We propose a construction SA1 by transforming each MAC-tag t_i into a matrix as follows: Let n be bit-length of MAC-tag and we separate a MAC-tag $t_i \in \{0,1\}^n$ into $(t_{i,1} \parallel t_{i,2} \parallel t_{i,3} \parallel t_{i,4}) \in (\{0,1\}^{\frac{n}{4}})^4$. Then, we regard each $t_{i,j} \in \{0,1\}^{\frac{n}{4}}$ $(1 \le j \le 4)$ as an element of the finite field $GF(2^{\frac{n}{4}})$, and set $T_i := \begin{bmatrix} t_{i,1} & t_{i,2} \\ t_{i,3} & t_{i,4} \end{bmatrix}$. Here, we note that such a matrix T_i is invertible with an overwhelming probability if the MAC meets pseudorandomness. Based on this transformation, SA1 = (SA1.SeqAgg, SA1.SAVrfy) is constructed in the following way.

- $\tau \leftarrow$ SA1.SeqAgg$(((id_{\ell_i}, t_i))_{i\in[\widehat{N}]})$: Generate an aggregate tag as follows:
  1. For each $i \in [\widehat{N}]$, let $T_i := \begin{bmatrix} t_{i,1} & t_{i,2} \\ t_{i,3} & t_{i,4} \end{bmatrix}$ be a matrix transformed from t_i as mentioned above.
  2. Output $\tau := T_1 T_2 \cdots T_{\widehat{N}}$.
- $1/0 \leftarrow$ SA1.SAVrfy(K, M, τ): For $K = \{(k_{id_i}, id_i)\}_{i\in[N]}$ and $M = ((m_i, id_{\ell_i}))_{i\in[\widehat{N}]}$, verify (M, τ) as follows:
  1. For each $i \in [\widehat{N}]$, compute $t_i' \leftarrow$ MAC.Tag$(k_{id_{\ell_i}}, m_i)$ and $\tau' \leftarrow$ SA1.SeqAgg$(((id_{\ell_i}, t_i'))_{i\in[\widehat{N}]})$.
  2. Output 1 if $\tau' = \tau$, or output 0 otherwise.

Then, we show the following lemma.

Lemma 1. *Given two aggregate tags* $\tau_1 \leftarrow$ SA1.SeqAgg$((id_{\ell_1}, t_1), \ldots, (id_{\ell_i}, t_i))$ *and* $\tau_2 \leftarrow$ SA1.SeqAgg$((id_{\ell_{i+1}}, t_{i+1}), \ldots, (id_{\ell_j}, t_j))$*, if MAC meets pseudorandomness, the probability that* $\tau_1\tau_2 = \tau_2\tau_1$ *holds is negligible.*

Proof. We denote the matrices τ_1 and τ_2 by

$$\tau_1 = \begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix}, \ \tau_2 = \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix},$$

where $a_i, b_i, c_i, d_i \in GF(2^{\frac{n}{4}})$ for $i \in \{1,2\}$. We have

$$\tau_1\tau_2 = \begin{bmatrix} a_1a_2 + b_1c_2 & a_1b_2 + b_1d_2 \\ a_2c_1 + c_2d_1 & b_2c_1 + d_1d_2 \end{bmatrix}, \ \tau_2\tau_1 = \begin{bmatrix} a_1a_2 + b_2c_1 & a_2b_1 + b_2d_1 \\ a_1c_2 + c_1d_2 & b_1c_2 + d_1d_2 \end{bmatrix}.$$

Then, $\tau_1\tau_2 = \tau_2\tau_1$ is equivalent to the conditions:

$$b_1c_2 = b_2c_1, \tag{1}$$
$$a_1b_2 + b_1d_2 = a_2b_1 + b_2d_1, \tag{2}$$
$$a_2c_1 + c_2d_1 = a_1c_2 + c_1d_2. \tag{3}$$

Hence, the number of the equations that must hold is three, while the number of variables a_i, b_i, c_i, d_i $(i = 1, 2)$ is eight. Therefore, if a_i, b_i, c_i, d_i $(i = 1, 2)$ are chosen uniformly at random, the probability that the Eqs. (1)–(3) hold is $(2^{-\frac{n}{4}})^3 = 2^{-\frac{3}{4}n}$. Therefore, if the MAC meets pseudorandomness, the probability that the Eqs. (1)–(3) hold is negligible. □

Construction 2. We construct SA2 = (SA2.SeqAgg, SA2.SAVrfy) by using hash functions in a simple way, and this construction is provably secure in the random oracle model. SA2 is given as follows: Let H be a random function $H : \{0,1\}^* \to \mathcal{T}$, where $\mathcal{T}$ is the tag space of a MAC scheme.

- $\tau \leftarrow$ SA2.SeqAgg$(((id_{\ell_i}, t_i))_{i\in[\widehat{N}]})$: Output $\tau := H(t_1, \ldots, t_{\widehat{N}})$.
- $1/0 \leftarrow$ SA2.SAVrfy(K, M, τ): Output 1 if $\tau = H(t'_1, \ldots, t'_{\widehat{N}})$, where $t'_i :=$ MAC.Tag$(k_{id_{\ell_i}}, m_i)$ for all $i \in [\widehat{N}]$, and output 0 otherwise.

By definition of random functions, we can see that the order of messages is guaranteed.

4.2 Security of Our Constructions

The following theorems show the security of our constructions.

Theorem 1. *If MAC meets pseudorandomness, SAMAC1 meets $(N-2)$-aggUF-CMA.*

Theorem 2. *If MAC meets UF-CMA, SAMAC2 meets $(N-2)$-aggUF-CMA.*

Proof of Theorem 1. We prove that SAMAC1 meets $(N-2)$-aggUF-CMA. Let $\mathcal{A}$ be a PPT adversary against SAMAC1. We define the following events:

- Succ: An event that $\mathcal{A}$ outputs a forgery breaking aggUF-CMA.
- New: An event that for a new message which is never queried, $\mathcal{A}$ makes a forgery against a MAC scheme which is not corrupted.
- Pre: An event that $\mathcal{A}$ makes a forgery against SAMACi without generating any forgeries against MACs.
- Cor: An event that $\mathcal{A}$ does not break any MAC schemes, but uses new MAC-tags generated by using corrupted ID's keys.
- Replace: An event that $\mathcal{A}$ replaces the sequence of messages queried to the tagging oracle.

Because the events New and Pre are exclusive, we have

$$\begin{aligned} Adv^{agg\text{-}uf}_{SAMAC,\mathcal{A}}(\lambda) &:= \Pr[\mathsf{Succ}] \leq \Pr[\mathsf{Succ} \wedge \mathsf{New}] + \Pr[\mathsf{Succ} \wedge \mathsf{Pre}] \\ &\leq \Pr[\mathsf{Succ} \wedge \mathsf{New}] + \Pr[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \mathsf{Replace}] \\ &\quad + \Pr[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \mathsf{Cor} \mid \overline{\mathsf{Replace}}] + \Pr[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \overline{\mathsf{Cor}} \mid \overline{\mathsf{Replace}}]. \end{aligned}$$

Therefore, it is sufficient to prove the following:

- $\Pr[\mathsf{Succ} \wedge \mathsf{New}] \leq \frac{N}{P_{inv}} \cdot Adv^{uf\text{-}cma}_{MAC,\mathcal{F}}(\lambda)$ for a function P_{inv} of N.
- $\Pr[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \mathsf{Replace}] \leq \varepsilon(\lambda)$ for a negligible function ε.
- $\Pr[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \mathsf{Cor} \mid \overline{\mathsf{Replace}}] \leq \frac{(N-C)}{P_{inv}} \cdot Adv^{pr}_{MAC,\mathcal{D}}(\lambda) + \frac{1}{2^n}$.
- $\Pr[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \overline{\mathsf{Cor}} \mid \overline{\mathsf{Replace}}] \leq \frac{(N-C)}{P_{inv}} \cdot Adv^{pr}_{MAC,\mathcal{D}}(\lambda) + \frac{1}{2^n}$.

Event $[\mathsf{Succ} \wedge \mathsf{New}]$: In this case, an adversary generates a forgery against a MAC scheme which the ID id fulfills $id \notin \mathcal{L}_{Cor}$. By using $\mathcal{A}$ breaking aggUF-CMA, we construct a PPT algorithm $\mathcal{F}$ breaking UF-CMA of MACs as follows.

Setup: Given the tagging oracle of a MAC, do the following.
1. Choose $id_i \in \mathcal{ID}$ for all $i \in [N]$,
2. Generate $k_{id_i} \leftarrow \mathrm{KGen}(1^\lambda, id_i)$ for all $i \in [N]$,
3. Set lists $\mathcal{L}_{Cor} = \emptyset$ and $\mathcal{L}_{SA} = \emptyset$.

Corrupt: When $\mathcal{A}$ submits an ID $id \in \mathcal{ID}$ to the oracle $\mathsf{Corrupt}(\cdot)$, return k_{id} and set $\mathcal{L}_{Cor} \leftarrow \mathcal{L}_{Cor} \cup \{id\}$. When $\mathcal{A}$ stops accessing $\mathsf{Corrupt}$ and moves to the $\mathsf{Tagging}$ phase, choose an ID $id^* \notin \mathcal{L}_{Cor}$ uniformly at random.

Tagging: For each query $M = ((m_i, id_{\ell_i}))_{i \in [\widehat{N}]}$ to the oracle $\mathsf{SATag}_K(\cdot)$ where $K := \{(k_{id_i}, id_i)\}_{i \in [N]}$, do the following for all $i \in [\widehat{N}]$.

- If $id_{\ell_i} \neq id^*$, compute $t_i = \mathrm{Tag}(k_{id_{\ell_i}}, m_i)$,
- If $id_{\ell_i} = id^*$, submit a message query m_i to the MAC oracle and receive the tag t_i.

  Return $\tau = \mathrm{SeqAgg}((id_{\ell_i}, t_i)_{i \in [\widehat{N}]})$ to $\mathcal{A}$ and set $\mathcal{L}_{SA} \leftarrow \mathcal{L}_{SA} \cup \{M\}$.

Output: When $\mathcal{A}$ outputs $M^* = ((m_i^*, id_{\ell_i^*}))_{i \in [\widetilde{N}]}$ and τ^*, do the following.
1. Move to the next step if the output of $\mathcal{A}$ meets the conditions of the security game except for $1 \leftarrow \mathrm{SAVrfy}(K, M^*, \tau^*)$, or abort this game otherwise.
2. For $i \in [\widetilde{N}]$ and id_{ℓ_i} except for id^*, compute $\tau_i^* = \mathrm{MAC.Tag}(k_{id_{\ell_i^*}}, m_i^*)$.
3. Let i^* be the order of the ID id^* in M^* and compute T_{i^*} in the following way: $T_{i^*} = T^{*-1}_{i^*-1} \cdot \cdots \cdot T^{*-1}_1 \cdot \tau \cdot T^{*-1}_{\widetilde{N}} \cdot \cdots \cdot T^{*-1}_{i^*+1} \in GF(2^n)$.
4. Recover a MAC-tag t_{i^*} from the matrix T_{i^*}.
5. Output (m_{i^*}, t_{i^*}) as a forgery of the MAC.

$\mathcal{F}$ simulates the environment of $\mathcal{A}$ completely. In Step 3 of Output phase, the probability that all matrices are invertible is at most $P_{inv} := (1 - \frac{1}{2^{n/4}})^{N-1}$. For all IDs $id_{\ell_i^*}$ $(i \in [\widetilde{N}])$ except for id^*, MAC tags t_i^* generated by using $k_{id_{\ell_i^*}}$ are valid. Therefore, t_{i^*} is also a valid MAC tag. Therefore, the success probability of $\mathcal{F}$ is at least $\frac{P_{inv}}{N} \cdot \Pr[\mathsf{Succ} \wedge \mathsf{New}]$.

Event $[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \mathsf{Replace}]$: From Lemma 1, $\Pr[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \mathsf{Replace}]$ is $2^{-\frac{3}{4}n}$ in SAMAC1.

Event $[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \mathsf{Cor} \mid \overline{\mathsf{Replace}}]$: In this case, we consider the following adversaries: We assume that for a query M, there exists a corrupted pair (id, m, t) between not corrupted pairs $(id_{i_1}, m_{i_1}, t_{i_1})$ and $(id_{i_2}, m_{i_2}, t_{i_2})$ such that i_1 is the first order among not corrupted IDs and i_2 is the last order among not corrupted IDs, in M. An adversary tries to replace the message/tag pair (m, t) with (m^*, t^*) such that $m^* \neq m$ and $t^* = \mathsf{Tag}(k_{id}, m^*)$. He cannot replace the message/tag pair without knowing t_{i_1} and t_{i_2}. We show that the probability that the event happens is negligible if MACs meet pseudorandomness.

Let Game-0 be the standard security game and let C be the number of corrupted IDs. For $X \in [N - C]$, we define Game-X where for one of IDs $id \notin \mathcal{L}_{Cor}$, the MAC's tagging algorithm is replaced with a random function $f_{id} : \mathcal{M} \to \mathcal{T}$. Then, we show that in Game-$(X-1)$ and Game-X, the difference between the success probabilities of them is negligible from pseudorandomness of MACs. We construct a PPT algorithm $\mathcal{D}$ breaking pseudorandomness of a MAC scheme. $\mathcal{D}$ can be constructed in the same way as in the above $\mathcal{F}$ except for the process of Output phase. We describe the process of $\mathcal{D}$ at Output phase as follows: When $\mathcal{A}$ outputs $M^* = ((m_i^*, id_{\ell_i^*}))_{i \in [\widetilde{N}]}$ and τ^*, do the following.

1. Move to the next step if the output of $\mathcal{A}$ meets the conditions of the security game except for $1 \leftarrow \mathrm{SAVrfy}(K, M^*, \tau^*)$, or abort this game otherwise.
2. For each $id_{\ell_i^*}$ $(i \in [\widetilde{N}])$ except for id^*, compute t_i^* by using the key $k_{id_{\ell_i^*}}$.
3. Compute the MAC-tag t_{id^*} of id^* from τ^* and the other tags computed in Step 2.
4. Submit m^* to the tagging oracle and receive the tag t.
5. Output 1 if $t_{id^*} = t$ and (m^*, id^*) has never been queried, or output 0 otherwise.

In Game-$(N - C)$, all outputs of f_{id} and MAC tags are hidden statistically. Therefore, the probability is at most $\frac{(N-C)}{P_{inv}} \cdot Adv^{pr}_{MAC,\mathcal{D}}(\lambda) + \frac{1}{2^n}$.

Event $[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \overline{\mathsf{Cor}} \mid \overline{\mathsf{Replace}}]$: In the same way as event $[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \mathsf{Cor} \mid \overline{\mathsf{Replace}}]$, we obtain $\Pr[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \overline{\mathsf{Cor}} \mid \overline{\mathsf{Replace}}] \leq \frac{(N-C)}{P_{inv}} \cdot Adv^{pr}_{MAC,\mathcal{D}}(\lambda) + \frac{1}{2^n}$.

From the discussion above, we have

$$\begin{aligned} Adv^{agg\text{-}uf}_{SAMAC1,\mathcal{A}}(\lambda) &\leq \frac{N}{P_{inv}} \cdot Adv^{uf\text{-}cma}_{MAC}(\lambda) + \frac{1}{2^{\frac{3}{4}n}} + 2\frac{(N-C)}{P_{inv}} \cdot Adv^{pr}_{MAC}(\lambda) + \frac{1}{2^{n-1}} \\ &\leq 3\frac{N}{P_{inv}} \cdot Adv^{pr}_{MAC}(\lambda) + \left(\frac{N}{P_{inv}} + 2\right)\frac{1}{2^n} + \frac{1}{2^{\frac{3}{4}n}}. \end{aligned}$$

We note that $Adv^{uf\text{-}cma}_{MAC}(\lambda) \leq Adv^{pr}_{MAC}(\lambda) + \frac{1}{2^n}$ holds (see [4]), and P_{inv} is $\left(1 - \frac{1}{2^{n/4}}\right)^{N-1}$. Therefore, the proof is completed. □

Proof of Theorem 2. Let $\mathcal{A}$ be a PPT adversary against SAMAC2. Let $\mathcal{L}_H$ be the list of the query/answer pairs of $H(\cdot)$. Let Forge be an event that $\mathcal{A}$ breaks SAMAC2 by making a forgery of the underlying MAC, and let Coll be an event that $\mathcal{A}$ finds a collision of the random oracle H. Then, we have $Adv^{agg\text{-}uf}_{SAMAC2,\mathcal{A}}(\lambda) := \Pr[\mathsf{Forge}] \leq \Pr[\mathsf{Coll}] + \Pr[\mathsf{Forge} \wedge \overline{\mathsf{Coll}}]$.

In the event Coll, an adversary tries to find a collision of H. We note that this case includes an attack that he replaces MAC-tags queried to H. The success probability is at most $\frac{Q_h^2}{2^{n+1}}$.

Next, we consider the event $[\mathsf{Forge} \wedge \overline{\mathsf{Coll}}]$. We construct a PPT algorithm $\mathcal{F}$ breaking UF-CMA as in the proof of Theorem 1 except for the process of Output phase. In this phase, when $\mathcal{A}$ outputs $((m_i, id_{\ell_i^*}))_{i\in[\widetilde{N}]}$ and τ^*, $\mathcal{F}$ does the following process.

1. Move to the next step if the output of $\mathcal{A}$ meets the conditions of the security game except for $1 \leftarrow \text{SAVrfy}(K, M^*, \tau^*)$, or abort this game otherwise.
2. Compute $t_i^* = \text{MAC.Tag}(k_{id_{\ell_i^*}}, m_i)$ except for id^*,
3. Find a pair $((t_i^*)_{i\in[\widetilde{N}]}, \tau^*)$ except for a tag of id^* from $\mathcal{L}_H$. Abort this game if there exists no such pair in $\mathcal{L}_H$.
4. Output the id^*'s pair (m^*, t^*).

The pair $(t_1^*, \ldots, t_{\widetilde{N}}^*, \tau^*)$ is in $\mathcal{L}_H$ with overwhelming probability because the probability that it outputs τ^* such that $\tau^* = H(t_1^*, \ldots, t_{\widetilde{N}}^*)$ is negligible without accessing to the random oracle $H(\cdot)$. Thus, $\mathcal{F}$'s output is a valid forgery breaking a MAC scheme. Therefore, we have $\Pr[\mathsf{Forge} \wedge \overline{\mathsf{Coll}}] \leq N \cdot Adv^{uf\text{-}cma}_{MAC,\mathcal{F}}(\lambda)$.

From the above, we obtain $Adv^{agg\text{-}uf}_{SAMAC2,\mathcal{A}}(\lambda) \leq N \cdot Adv^{uf\text{-}cma}_{MAC,\mathcal{F}}(\lambda) + \frac{Q_h^2}{2^{n+1}}$, and the proof is completed. □

5 Application: Sending Long Data by Data-Partitioning

Suppose that a device wants to send a long message in a wireless network, but the message is too long to directly transmit because of restrictions in the network. In this case, we usually utilize a data partitioning method to transmit the long data: We first divide a long message M into (at most) N pieces $m_1, m_2, \ldots, m_N$ (e.g., each piece may be called a packet); For each divided part m_j $(1 \leq j \leq N)$, the device generates a MAC tag $t_j \leftarrow$Tag$(k, (m_j, j))$; The device sends $((m_j, j), t_j)$ for $j = 1, 2, \ldots, N$ by possibly different paths in the network; A receiver obtains $\{(m_j, t_j)\}_{j\in[N]}$, where we assume that divided parts $m_1, m_2, \ldots, m_N$ do not necessarily reach with the correct order (e.g., some of which may delayed in the network) and he will check the validity of both divided data and their ordering to correctly recover the message M. In this situation, we note that N tags are transmitted in the wireless network, which may cause a traffic problem if there are an enormous number of devices connected to the network and each device wants to send a long message. Our idea is to apply a sequential aggregate MAC under consideration in the previous sections in order to reduce the numbers of tags for divided data, so that we resolve the problem by reducing the amount of tags with the aggregation technique without changing the structure of the underlying MACs.

Formally, suppose that an existing authentication protocol utilizes a MAC scheme MAC=(KGen, Tag, Vrfy) as follows, where a secret key $k \leftarrow$KGen(1^λ) is already generated and installed in a device and such secret keys are generally different in devices:

- Transmission by Data-Partitioning:
  1. For a long message M, generate divided messages $(m_1, 1), (m_2, 2), \dots, (m_N, N)$ from M by a data partitioning technique.
  2. For each (m_j, j), generate its tag $t_j \leftarrow$Tag$(k, (m_j, j))$.
  3. It transmits $((m_j, j), t_j)$ for $j = 1, 2, \dots, N$ by possibly different paths in the network.
- Verification: On receiving $\{((m_j, j), t_j)\}_{j\in[N]}$, it checks both the validity of both divided data and their ordering: If $1 \leftarrow$Vrfy$(K, (m_j, j), t_j)$ for every $j \in [N]$, M is recovered by the sequential data $(m_1, m_2, \dots, m_N)$; otherwise, it rejects the data.

In order to resolve a traffic problem, we consider to embed a SeqAgg algorithm into a device and a SAVrfy algorithm into an verification protocol/system as an application of our sequential MACs. Then, we propose the following:

- Transmission by Data-Partitioning:
  1 and 2. The same in the above protocol.
  3. Compute $\tau \leftarrow$SeqAgg$(((t_i, i))_{i\in[N]})$, and then transmit N pieces $((m_1, 1), T)$, $(m_2, 2), \dots, (m_N, N)$ by possibly different paths in the network, where we note that a tag is attached only to $(m_1, 1)$.
- Verification: On receiving $((m_1, 1), T)$ and $\{((m_j, j), t_j)\}_{2\le j\le N}$, it checks both the validity of both divided data and their ordering: If $1 \leftarrow$SAVrfy$(k, \tilde{M}, \tau)$ where $\tilde{M} = ((m_1, 1), (m_2, 2), \dots, (m_N, N))$, M is recovered by the sequential data $(m_1, m_2, \dots, m_N)$; otherwise, it rejects the data.

Here, we note that in each device, the same key k is used for generating N tags $t_1, t_2, \dots, t_N$. Therefore, if a device keeps the key secure, it is sufficient to apply C-aggUF-CMA secure sequential aggregate MACs with $C = 0$.

6 Our Construction of HF Sequential Aggregate MAC

We construct a partial invertible MAC scheme meeting computational almost universal from MAC schemes. By applying this construction to Construction 6.10 of [5], we can obtain a history-free (hf) sequential aggregate MAC scheme.

Let MAC = (KGen, Tag, Vrfy) be a MAC scheme. First, we define the following property of the MAC.

Partial Inversion. MAC meets partial inversion if there exists the following polynomial time algorithm: For any secret key k and any message m, and a tag τ given as input, the algorithm returns m' such that $\tau \leftarrow \mathrm{Tag}(k, (m \parallel m'))$ for some $m' \in \{0,1\}^{\mathrm{poly}(\lambda)}$.

Next, we construct a scheme meeting partial inversion and pseudorandomness from our SA1. Let MAC = (KGen, Tag, Vrfy) be the underlying MAC. Then, PIMAC = (PIMAC.KGen, PIMAC.Tag, PIMAC.Vrfy) is constructed as follows.

- $k \leftarrow \mathrm{PIMAC.KGen}(1^\lambda)$: Output $k \leftarrow \mathrm{KGen}(1^\lambda)$.
- $\tau \leftarrow \mathrm{PIMAC.Tag}(k, m \parallel \tau')$: On input a secret key k and a message $(m \parallel \tau') \in \mathcal{M} \times \mathcal{T}_{agg}$ where any $\tau' \in \mathcal{T}_{agg}$ is a matrix of SA1, generate a tag τ in the following way.
  1. Compute $t \leftarrow \mathrm{Tag}(k, m)$ and let T be a matrix for t based on SA1.
  2. Output $\tau := T \cdot \tau' \cdot T \in \mathcal{T}_{agg}$.
- $1/0 \leftarrow \mathrm{PIMAC.Vrfy}(k, m \parallel \tau', \tau)$: On input a secret key k, a message $m \parallel \tau'$, and a MAC tag τ, verify the message/tag pair $(m \parallel \tau', \tau)$ as follows.
  1. Compute $\bar{\tau} \leftarrow \mathrm{PIMAC.Tag}(k, m \| \tau')$.
  2. Output 1 if $\bar{\tau} = \tau$, or output 0 otherwise.

Then, we show the following lemma.

Lemma 2. *PIMAC meets partial inversion. Furthermore, if MAC meets pseudorandomness, PIMAC also meets pseudorandomness.*

Proof. First, we prove that PIMAC meets partial inversion by constructing the following partial inversion algorithm: It takes a secret key k, a message m, and τ as input, and does the following.

1. Compute $t \leftarrow \mathrm{Tag}(k, m)$ and let T be a matrix transformed from t.
2. Output $\tau' := T^{-1} \cdot \tau \cdot T^{-1} \in \mathcal{T}_{agg}$.

Then, we can see that the output τ' is valid.

Second, we prove that PIMAC meets pseudorandomness. Let $\mathcal{A}$ be a PPT adversary breaking the pseudorandomness of PIMAC. We construct a PPT algorithm $\mathcal{B}$ breaking the pseudorandomness of the underlying MAC as follows: It is given the oracle of a MAC or a random function. When $\mathcal{A}$ submits a message query $m \| \tau'$, it submits m to the given oracle and receives the value t. Then, it computes τ following PIMAC.Tag algorithm and returns it. When $\mathcal{A}$ outputs the guessing bit $b' \in \{0,1\}$, $\mathcal{B}$ also outputs b'.

If $\mathcal{A}$ breaks the pseudorandomness of PIMAC, $\mathcal{B}$ also breaks the pseudorandomness of MAC. This completes the proof. $\square$

Let HF-SAMAC be a sequential aggregate MAC obtained by applying PIMAC to Construction 6.10 of [5]. Then, by Theorem 6.11 of [5], we have:

Proposition 2. *If PIMAC meets pseudorandomness and partial inversion, HF-SAMAC meets the aggregate unforgeability of Definition 5.4 in [5].*

7 Conclusion

In this paper, we introduced a new model of sequential aggregate MACs where an aggregation algorithm generates a sequential aggregate tag depending only on multiple and independent MAC tags without any secret-key, and we formally defined security in this model. Our model and security are quite different from those of previous works [5,6,9]. In addition, we proposed two generic constructions, SAMAC1 and SAMAC2, starting from any MACs, with formal security proofs. And, we compared the existing ones and ours in terms of universal applicability, security, and efficiency. As a result, SAMAC2 is superior to others from all aspects of evaluation items, though the security proof is given in the random oracle model. It is interesting to consider SAMAC1 as well in the standard model, and it can be transformed into a history-free sequential aggregate MAC in the model of [5] without changing the input-formats of MACs or adding any other primitives except for MACs.

We note that, if a sequence of messages are rejected in our sequential aggregate MACs, we cannot identify which message has been invalid (e.g., some of them was forged, or their order was wrong). Hirose and Shikata [7] recently proposed (non-sequential) aggregate MACs in which we could identify which message was invalid, if a set of messages are rejected in their aggregate MACs. Our future work includes extension of [7] for sequential aggregate MACs.

Acknowledgements. The authors would like to thank the anonymous referees for their helpful comments. This research was conducted under a contract of Research and Development for Expansion of Radio Wave Resources funded by the Ministry of Internal Affairs and Communications, Japan.

References

1. NIST Special Publication 800–38G: Recommendation for block cipher modes of operation: the CMAC mode for authentication. National Institute of Standards and Technology (2005)
2. Bellare, M.: New proofs for NMAC and HMAC: security without collision resistance. J. Cryptol. **28**(4), 844–878 (2015)
3. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_1
4. Bellare, M., Kilian, J., Rogaway, P.: The security of the cipher block chaining message authentication code. J. Comput. Syst. Sci. **61**(3), 362–399 (2000)
5. Eikemeier, O., et al.: History-free aggregate message authentication codes. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 309–328. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15317-4_20
6. Hirose, S., Kuwakado, H.: Forward-secure sequential aggregate message authentication revisited. In: Chow, S.S.M., Liu, J.K., Hui, L.C.K., Yiu, S.M. (eds.) ProvSec 2014. LNCS, vol. 8782, pp. 87–102. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12475-9_7

7. Hirose, S., Shikata, J.: Non-adaptive group-testing aggregate MAC schemes. In: ISPEC. Lecture Notes in Computer Science, Springer (2018, to appear). Available at Cryptology ePrint Archive Report 2018/448
8. Katz, J., Lindell, A.Y.: Aggregate message authentication codes. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 155–169. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79263-5_10
9. Ma, D., Tsudik, G.: Extended abstract: forward-secure sequential aggregate authentication. In: IEEE Symposium on Security and Privacy, pp. 86–91. IEEE Computer Society (2007)
10. Tomita, S., Watanabe, Y., Shikata, J.: Sequential aggregate authentication codes with information theoretic security. In: CISS, pp. 192–197. IEEE (2016)