# A Note on Aggregate MAC Schemes

Shoichi Hirose[1] Junji Shikata[2]

[1]University of Fukui, Japan

[2]Yokohama National University, Japan

13/11/2018
ASK 2018, Kolkata

# Introduction

Message authentication code (MAC)

| Sender | | Receiver |
|---|---|---|
| $t_i = F_K(M_i)$ | $\xrightarrow{(M_1, t_1)}$ $\xrightarrow{(M_2, t_2)}$ $\vdots$ | $t_i = F_K(M_i)$ ? |

Aggregate MAC [Katz, Lindell 2008]

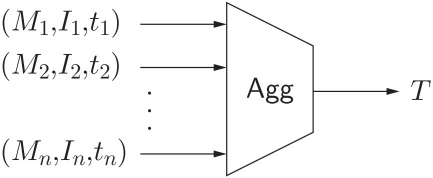
- Inspired by aggregate signature
- Generate an aggregate tag for multiple messages

$$T \leftarrow \mathsf{Aggregate}((M_1, I_1, t_1), \ldots, (M_n, I_n, t_n))$$

- Check the validity of messages in a single verification w.r.t. $T$
- Reduce the amount of storage and/or communication

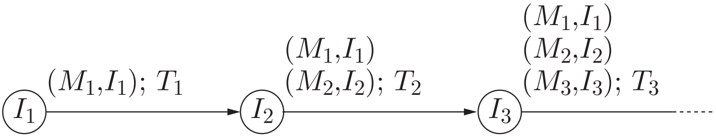
# Two Flavours of Aggregation

(Non-sequential) aggregation: The order does not matter




Often $T \leftarrow \mathsf{Agg}(t_1, t_2, \ldots, t_n)$

Sequential aggregation: The order matters




Called history-free if $T_j \leftarrow \mathsf{SeqAgg}_{K_j}(M_j, I_j, T_{j-1})$

# Brief Overview

Topics of This Talk

- Application of non-adaptive group-testing to aggregate MAC
- Sequential aggregate MAC

Related Work

- (Non-sequential) Aggregate MAC
  - Katz, Lindell (2008)
- Sequential aggregate MAC
  - Eikemeier, Fischlin, et al. (2010)
- Forward-secure sequential aggregate MAC (for secure logging)
  - Schneier and Kelsey (1999)
  - Ma and Tsudik (2007)
  - Hirose and Kuwakado (2014)

1. Non-adaptive Group Testing Aggregate MAC

2. Sequential Aggregate MAC

# Motivation

Aggregate MAC

- Generate an aggregate tag for multiple messages

$$T \leftarrow \mathsf{Aggregate}((M_1, I_1, t_1), \ldots, (M_n, I_n, t_n))$$

- Check the validity of messages in a single verification w.r.t. $T$
  - If valid, all messages are OK.
  - Otherwise, some are invalid, but we can't see which.

Problem: Identify the invalid messages with fewer than $n$ agg. tags

Our solution: Apply group testing to aggregate MAC

Two types of group testing

- Non-adaptive: All tests are chosen in advance
- Adaptive: A new test can be chosen after the current test

# Non-adaptive Group Testing

Specified by a binary matrix (Group-testing matrix):

$$
\begin{array}{cc}
 & \begin{array}{cccc} \text{s1} & \text{s2} & \text{s3} & \text{s4} \end{array} \\
\begin{array}{c} \text{test1} \\ \text{test2} \\ \text{test3} \end{array} & \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}
\end{array}
$$

- s1, s2, s3, and s4 are samples.
- Each sample is either negative or positive.
- The result of a test is
  - negative $\iff$ All the involved samples are negative
  - positive $\iff$ Some of the involved samples are positive
- Identify the positive samples with $(\#$ of tests$) < (\#$ of samples$)$
  Assumption: $\#$ of positive samples is upper-bounded

# $d$-disjunct GT Matrix

**Definition (GT matrix $\boldsymbol{G}$ is $d$-disjunct)**

For any $(d+1)$ columns $\boldsymbol{g}_{j_1}, \boldsymbol{g}_{j_2}, \ldots, \boldsymbol{g}_{j_{d+1}}$, there exists some $i$ s.t.

- $i$-th coordinate of $\boldsymbol{g}_{j_1} \vee \boldsymbol{g}_{j_2} \vee \cdots \vee \boldsymbol{g}_{j_d}$ is $0$
- $i$-th coordinate of $\boldsymbol{g}_{j_{d+1}}$ is $1$

$d$-disjunctness guarantees: $(\#$ of positive samples$) \leq d \implies$

each negative sample is included in a test only with negative samples

Non-adaptive group testing based on $d$-disjunct GT matrix

- identifies all the positive samples if $(\#$ of them$) \leq d$
- All samples involved in negative tests are negative.
- All the remaining samples are positive.

# Agenda

- Syntax
- Security requirements
  - Unforgeability
  - Identifiability: Completeness and soundness
- Generic construction
- Two instantiations
- Analysis of provable security

# Related Work

Agregate MAC for multiple users [Katz-Lindell 08]

- Formalized the syntax and security requirement
- Proposed scheme: For $(M_1, I_1), (M_2, I_2), \ldots, (M_n, I_n)$,
  - $t_j = \mathsf{MAC}(K_j, M_j)$
  - The aggregate tag is $T = t_1 \oplus t_2 \oplus \cdots \oplus t_n$
- Proved the security

Application of group-testing to MAC [Goodrich et al. 05], [Minematsu 15]

- Both of them assumes a single-user setting
- Tag aggregate requires a secret key

# Aggregate MAC: Syntax

Aggregate MAC (AM) consists of the following algorithms:

Key generation $K \leftarrow \mathsf{KG}(1^p)$

- $p$ is a security parameter

Tagging $t \leftarrow \mathsf{Tag}(K_I, M, I)$

Aggregate $T \leftarrow \mathsf{Agg}((M_1, I_1, t_1), \ldots, (M_n, I_n, t_n))$

- Secret keys are not used
- Often $T \leftarrow \mathsf{Agg}(t_1, \ldots, t_n)$

Verification $d \leftarrow \mathsf{Ver}((K_1, \ldots, K_n), ((M_1, I_1), \ldots, (M_n, I_n)), T)$

- The decision $d$ is either $\top$ (valid) or $\perp$ (invalid)

# Aggregate MAC: Security Requirement

The security requirement is unforgeability

An adversary $\mathbf{A}$ against AM is given access to the following oracles:

Tagging receives $(M, I)$ and returns tag $t \leftarrow \mathsf{Tag}(K_I, M, I)$

Corrupt receives $I$ and returns $K_I$

Verification receives $(((M_1, I_1), \ldots, (M_n, I_n)), T)$ and returns $d \in \{\top, \bot\}$

$$\mathrm{Adv}^{\mathrm{uf}}_{\mathsf{AM}}(\mathbf{A}) \triangleq \Pr[\mathbf{A} \text{ succeeds in forgery}]$$

$\mathrm{Adv}^{\mathrm{uf}}_{\mathsf{AM}}(\mathbf{A})$ should be negligibly small for any efficient $\mathbf{A}$

$\mathbf{A}$ succeeds in forgery if $\mathbf{A}$ asks $Q = (((M_1, I_1), \ldots, (M_n, I_n)), T)$ to $\mathcal{VO}$ satisfying the following conditions:

- $Q$ is judged valid
- $\mathbf{A}$ asks neither $(M_j, I_j)$ to $\mathcal{TO}$ nor $I_j$ to $\mathcal{CO}$ for $\exists j$ before $Q$

# Group-Testing Aggregate (GTA) MAC

GTA MAC scheme using a $u \times n$ group-testing matrix

Key generation $K \leftarrow \mathsf{KG}(1^p)$

Tagging $t \leftarrow \mathsf{Tag}(K_I, M, I)$

Group-testing aggre $(T_1, \ldots, T_u) \leftarrow \mathsf{GTA}((M_1, I_1, t_1), \ldots, (M_n, I_n, t_n))$

- Secret keys are not used
- An aggregate tag is produced for each test

Group-testing verif

$J \leftarrow \mathsf{GTV}((K_1, \ldots, K_n), ((M_1, I_1), \ldots, (M_n, I_n)), (T_1, \ldots, T_u))$

- $J$ is a set of $(M_{j'}, I_{j'})$'s judged invalid

Security requirements

- Unforgeability
- Identifiability
  - Completeness: GTV judges any valid $(M, I, t)$ to be valid
  - Soundness: GTV judges any invalid $(M, I, t)$ to be invalid

# Unforgeability (1/2)

An adversary $\mathbf{A}$ against GTAM is given access to the oracles:

Tagging receives $(M, I)$ and returns $t \leftarrow \mathsf{Tag}(K_I, M, I)$

Corrupt receives $I$ and returns $K_I$

Group-testing verification

receives $(((M_1, I_1), \ldots, (M_n, I_n)), (T_1, \ldots, T_u))$ and returns the set of invalid $(M_j, I_j)$'s $J$

The advantage of $\mathbf{A}$ against GTAM w.r.t. unforgeability

$$\mathrm{Adv}^{\mathrm{uf}}_{\mathsf{GTAM}}(\mathbf{A}) \triangleq \Pr[\mathbf{A} \text{ succeeds in forgery}]$$

$\mathrm{Adv}^{\mathrm{uf}}_{\mathsf{GTAM}}(\mathbf{A})$ should be negligibly small for any efficient $\mathbf{A}$

## Unforgeability (2/2)

$\mathbf{A}$ succeeds in forgery if $\mathbf{A}$ asks $\mathcal{GTVO}$ a query

$$Q = (((M_1, I_1), \ldots, (M_n, I_n)), (T_1, \ldots, T_u))$$

satisfying that there exists some $(M_j, I_j)$ s.t.

- $(M_j, I_j)$ is judged valid by $\mathcal{GTVO}$
- $\mathbf{A}$ asks neither $(M_j,\ I_j)$ to $\mathcal{TO}$ nor $I_j$ to $\mathcal{CO}$ before asking $Q$

## Identifiability: Completeness and Soundness

An adversary $\mathbf{A}$ is given access to the following oracles:

Tagging receives $(M, I)$ and returns $t \leftarrow \mathsf{Tag}(K_I, M, I)$

Corrupt receives $I$ and returns $K_I$

Group-testing receives $Q = ((M_1, I_1, t_1), \ldots, (M_n, I_n, t_n))$

1. applies group testing to $Q$
2. returns the result

The advantage of $\mathbf{A}$ against GTAM w.r.t.

- completeness

$$\mathrm{Adv}^{\text{id-c}}_{\mathsf{GTAM}}(\mathbf{A}) \triangleq \Pr\left[\mathcal{GTO} \text{ judges some valid } (M_j, I_j, t_j) \text{ invalid}\right]$$

- soundness

$$\mathrm{Adv}^{\text{id-s}}_{\mathsf{GTAM}}(\mathbf{A}) \triangleq \Pr\left[\mathcal{GTO} \text{ judges some invalid } (M_j, I_j, t_j) \text{ valid}\right]$$

Both advantages should be negligibly small for any efficient $\mathbf{A}$

# Generic Construction

Generic GTA MAC using

- Aggre MAC $\mathsf{AM} = (\mathsf{KG}, \mathsf{Tag}, \mathsf{Agg}, \mathsf{Ver})$
- GT matrix $\boldsymbol{G}$

Key generation KG

Tagging Tag

Group-testing aggre $(T_1, \ldots, T_u) \leftarrow \mathsf{GTA}(t_1, \ldots, t_n)$

$$\begin{array}{l} T_1 \leftarrow \mathsf{Agg}(t_1, t_2) \\ T_2 \leftarrow \mathsf{Agg}(t_1, t_3) \\ T_3 \leftarrow \mathsf{Agg}(t_2, t_3, t_4) \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

Group-testing verif For $(((M_1, I_1), \ldots, (M_n, I_n)), (T_1, \ldots, T_u))$,

1. $t'_j \leftarrow \mathsf{Tag}(K_j, M_j, I_j)$ for $1 \leq j \leq n$
2. $(T'_1, \ldots, T'_u) \leftarrow \mathsf{GTA}(t'_1, \ldots, t'_n)$
3. For $1 \leq i \leq u$, if $T_i = T'_i$, all the involved $(M_j, I_j)$'s are valid
4. Remaining $(M_j, I_j)$'s are invalid

# Unforgeability of Generic Construction

Generic GTA MAC is UF $\Longleftarrow$ Underlying Aggre MAC is UF

## Theorem

*For any* $\mathbf{A}$ *against* GTAM, *there exists some* $\mathbf{B}$ *against* AM *s.t.*

$$\mathrm{Adv}^{\mathrm{uf}}_{\mathsf{GTAM}_{\mathrm{g}}}(\mathbf{A}) \leq \mathrm{Adv}^{\mathrm{uf}}_{\mathsf{AM}}(\mathbf{B})$$

| | $\mathbf{A}$ | $\mathbf{B}$ |
|---|---|---|
| *Run time* | $\leq s$ | $\leq s$ |
| *Tagging queries* | $\leq q_{\mathrm{t}}$ | $\leq q_{\mathrm{t}}$ |
| *Corrupt queries* | $\leq q_{\mathrm{c}}$ | $\leq q_{\mathrm{c}}$ |
| *Verif queries* | $\leq q_{\mathrm{v}}$ | $\leq uq_{\mathrm{v}}$ |

## Identifiability of Generic Construction

Generic GTA MAC satisfies completeness $\Longleftarrow$

- GTA matrix is $d$-disjunct
- Each query to $\mathcal{GTO}$ contains at most $d$ invalid $(M_j, I_j, t_j)$'s

**Theorem (Completeness)**

$$\mathrm{Adv}^{\text{id-c}}_{\mathsf{GTAM}_\mathrm{g}}(\mathbf{A}) = 0$$

Generic GTA MAC does not necessarily satisfy soundness

- Unforgeability guarantees weak soundness

## Two Instantiations

Two instantiations for group-testing aggregate:

- Based on Katz-Lindell AMAC: $T \leftarrow t_1 \oplus t_2 \oplus \cdots \oplus t_n$
- Based on cryptographic hashing: $T \leftarrow H(t_1, t_2, \ldots, t_n)$

Security

- Both satisfy unforgeability and completeness
- For soundness:
  - GTA MAC based on Katz-Lindell does not satisfy soundness

    Eg.) Let $(M_1, I_1, t_1)$ and $(M_2, I_2, t_2)$ be valid tuples

    The group test for invalid tuples

    $$(M_1, I_1, t_1 \oplus c) \text{ and } (M_2, I_2, t_2 \oplus c)$$

    gets valid since $t_1 \oplus t_2 = (t_1 \oplus c) \oplus (t_2 \oplus c)$
  - GTA MAC using hashing for aggregate satisfies soundness

    $\Longleftarrow$ $H$ is a random oracle

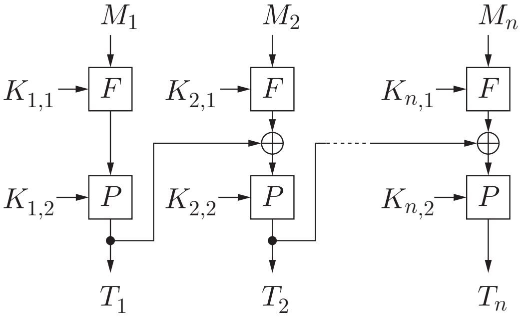
1. Non-adaptive Group Testing Aggregate MAC

2. Sequential Aggregate MAC

## Motivation

[Eikemeier, Fischlin, et al. 2010] proposed two schemes:

1. Using CMAC
2. Generic scheme using PRF $F$ and PRP $P$




Our question:

- PRP is indispensable?
- Simpler construction?

# Syntax

Sequential aggregate MAC (SAM) consists of the following algorithms:

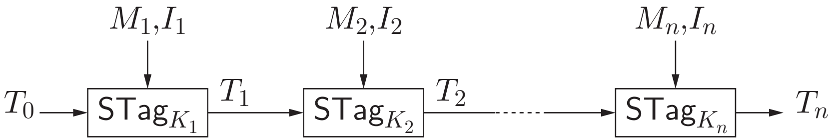
Key generation $K \leftarrow \mathsf{KG}(1^p)$

Sequential Aggregate Tagging $T \leftarrow \mathsf{STag}(K_I, M, I, T')$

- $T'$ is called an aggregate-so-far tag

Verification $d \leftarrow \mathsf{SVer}((K_1, \ldots, K_n), ((M_1, I_1), \ldots, (M_n, I_n)), T_n)$

- Decision $d \in \{\top, \perp\}$



# Security Requirement (1/2)

The security requirement of SAM is unforgeability

An adversary $\mathbf{A}$ against SAM is given access to the following oracles:

Seq agg tagging returns aggregate tag $T$ for query $(M, I), T'$

Corrupt returns $K_I$ for query $I$

Verification returns $d \in \{\top, \bot\}$ for query $((M_1, I_1), \ldots, (M_n, I_n)), T_n$

$\mathbf{A}$ is allowed to make multiple queries adaptively to each oracle

The advantage of $\mathbf{A}$ against SAM is

$$\mathrm{Adv}^{\mathrm{uf}}_{\mathsf{SAM}}(\mathbf{A}) \triangleq \Pr[\mathbf{A} \text{ succeeds in forgery}]$$

# Security Requirement (2/2)

$\mathbf{A}$ succeeds in forgery if $\mathbf{A}$ asks the verification oracle a query

$$Q = (((M_1, I_1), \ldots, (M_n, I_n)), T_n)$$

satisfying the following conditions:

- $Q$ is judged valid
- There exists some $j \in [1, n]$ s.t.
  - $\mathbf{A}$ does not ask $(M_j, I_j, T'_{j-1})$ to the seq agg tagging oracle
  - $\mathbf{A}$ does not ask $I_j$ to the corrupt oracle

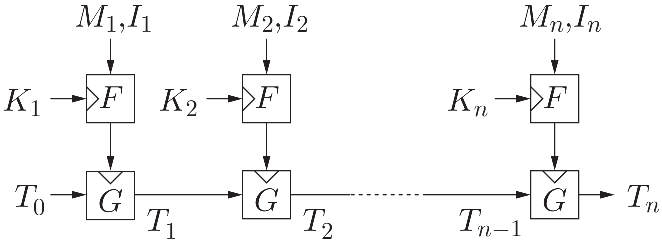
  before $Q$

# The First Proposed Scheme

Using PRF $F$ and PRP $G$

- Suitable for a block cipher

Sequential Aggregate Tagging $T_i = G_{F_{K_i}(M_i,I_i)}(T_{i-1})$




Uses the “tag” of a message by $F$ as a secret key of $G$ for aggregate

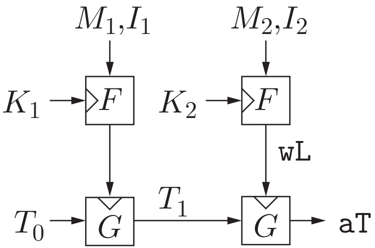
# $G$ Should Be a PRP (1/2)

Suppose that $G$ is a secure PRF with a weak key $\mathtt{wL}$ s.t.

$$G_{\mathtt{wL}}(T) = \mathtt{aT} \text{ for any } T$$

Then, the following attack always succeeds in forgery:

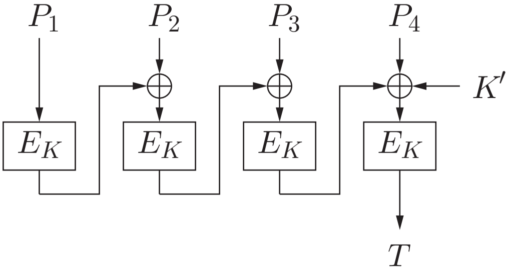
1. Ask $I_2$ to the corrupt oracle and obtain $K_2$.
2. Compute $M_2$ s.t. $F_{K_2}(M_2, I_2) = \mathtt{wL}$.
3. $(((M_1, I_1), (M_2, I_2)), \mathtt{aT})$ is a successful forgery for any $(M_1, I_1)$



# $G$ Should Be a PRP (2/2)

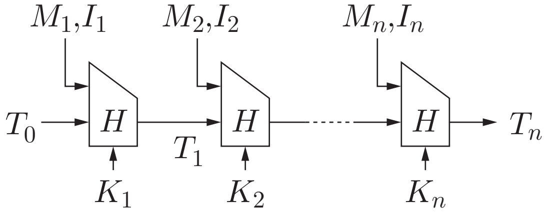
With knowledge of $K_2$, it is easy to compute $(M_2, I_2) = F^{-1}_{K_2}(\mathtt{wL})$

- if $F$ is a block cipher
- if $F$ is CMAC



# The Second Proposed Scheme (Naive Scheme)

Sequential Aggregate Tagging $T_i = H_{K_i}(T_{i-1}, M_i, I_i)$



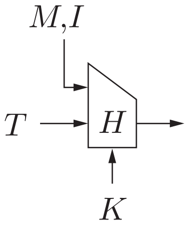

Question: Security requirement for $H$?

- Notice that $K_i$'s can be corrupted

# Security requirement for $H$ (1/2)

Sufficient conditions:

- $H$ keyed via $K$ is PRF, and
- $H$ keyed via $T$ is PRF under some leakage of $T$ due to verification

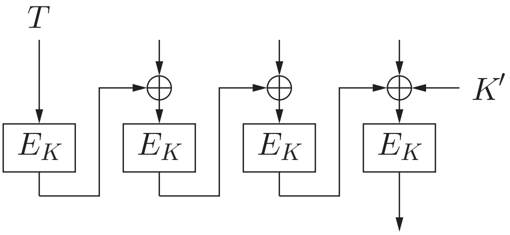



- CMAC does not satisfy the requirement
- HMAC seems OK

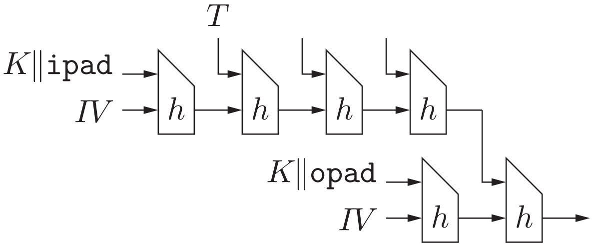
The naive scheme may be suitable for a hash function

# Security requirement for $H$ (2/2)

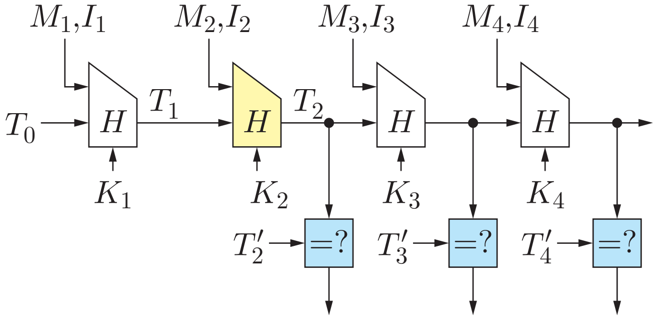
CMAC is not PRF if keyed via $T$




HMAC



# Intuitive Idea of Unforgeability Proof




- $(M_2, I_2, T_1)$ is new and $K_2$ is not corrupted $\implies$ $T_2$ is random
- Verification only leaks equality to given $T'_j$

# Conclusion

Application of Non-adaptive group-testing to aggregate MAC

- Formalization of syntax and security requirements
- Generic construction and two instantiations

Sequential aggregate MAC

- A scheme for a block cipher
- A scheme for a hash function

Other work

- Application of adaptive group-testing to aggregate MAC

Future work

- Efficient verification algorithm of $d$-disjunctness of GT matrix
- Security analysis of the naive scheme using CMAC for seq agg MAC