# Forward Secure Message Franking

Hiroki Yamamuro[1(✉)], Keisuke Hara[1,2], Masayuki Tezuka[1], Yusuke Yoshida[1], and Keisuke Tanaka[1]

[1] Tokyo Institute of Technology, Tokyo, Japan
yamamuro.h.ab@m.titech.ac.jp
[2] National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan

**Abstract.** Message franking is introduced by Facebook in end-to-end encrypted messaging services. It allows to produce verifiable reports of malicious messages by including cryptographic proofs generated by Facebook. Recently, Grubbs et al. (CRYPTO'17) proceeded with the formal study of message franking and introduced committing authenticated encryption with associated data (CAEAD) as a core primitive for obtaining message franking.
In this work, we aim to enhance the security of message franking and propose forward security for message franking. It guarantees the security associated with the past keys even if the current keys are exposed. Firstly, we propose the notion of key-evolving message franking including additional key update algorithms. Then, we formalize forward security for five security requirements: confidentiality, ciphertext integrity, unforgeability, receiver binding, and sender binding. Finally, we show a construction of forward secure message franking based on CAEAD, forward secure pseudorandom generator, and forward secure message authentication code.

**Keywords:** message franking · forward security · abusive verifiable reports

## 1 Introduction

### 1.1 Background

*Message Franking.* Billions of people use messaging services such as WhatsApp [22], Signal [19], and Facebook Messenger [11]. In these services, the security goal is end-to-end security: the third party including the service providers cannot compromise the security of messages. Keeping the messages secret from the service providers has recently led to the following problem: when the receiver receives malicious messages such as spam, phishing links, and so on, he/she attempts to report them to the service providers so that they could take measures against the sender. However, the service providers are not able to judge the reported messages were actually sent by the particular sender.

To tackle this problem, Facebook introduced the notion of message franking [12]. In message franking, the sender generates a commitment to a message, called a franking tag, along with a ciphertext generated using a secret key (which is shared with the receiver) and sends them to Facebook. Next, Facebook generates a cryptographic proof, called a reporting tag, from the franking tag using a tagging key (which is held by Facebook) and gives the ciphertext, the franking tag, and the reporting tag to the receiver. Then, the receiver decrypts the ciphertext using the secret key, gets the message and an opening of the franking tag, and verifies the validity of the franking tag. If the receiver wants to report a malicious message, he/she sends the reporting tag and the opening to Facebook in addition to the message. It enables Facebook to verify the specific sender actually sent the reported messages by validating the reporting tag using the tagging key.

Grubbs, Lu, and Ristenpart [13] initiated the formal study of message franking. They introduced committing authenticated encryption with associated data (CAEAD) as a core primitive for message franking. CAEAD is authenticated encryption with associated data (AEAD) that uses a small part of the ciphertext as a commitment to the message.

*Forward Security.* In general, most of cryptographic primitives are designed to be secure as long as their secret keys are not compromised. Thus, the exposure of secret keys is the greatest threat for many cryptographic schemes. Especially, if secret keys are compromised, the security of schemes is at risk not only after compromising but also in the past.

Forward security [9, 14] is one of the major solutions to address the exposure of secret keys, which ensures that compromise of the secret keys in the present does not influence the security of ciphertexts and tags generated in the past. Roughly, considering forward security, the lifetime of the system is divided into $n$ time periods and secret keys are updated in each period so that any past secret keys cannot be calculated from the current secret keys. To date, forward security was defined in a digital signature scheme [3], a symmetric encryption scheme [4], and an asymmetric encryption scheme [5]. Recently, the definition of forward security has also been considered for practical cryptographic primitives, such as a non-interactive key exchange scheme [20], a 0-RTT key exchange protocol [8, 15], a 0-RTT session resumption protocol [2], and Signal protocol [1, 7].

In a message franking scheme, the exposure of secret keys causes malicious adversaries to decrypt and tamper with the past ciphertexts and forge the past reporting tags. Moreover, it also enables them to falsely report messages that were actually not sent and generate messages that fail verification. To avoid these problems, the challenge of achieving forward security in a message franking scheme is important.

### 1.2 Our Contribution

In this paper, we initiate the study on forward security for message franking. For capturing forward security, we firstly formalize key-evolving message franking, which includes two key update algorithms for a secret key and a tagging key, respectively.

Roughly, forward security in message franking guarantees the security of the ciphertexts and reporting tags generated with the past keys, even if the current keys are exposed. More precisely, we define forward security on key-evolving message franking for five security requirements: confidentiality, ciphertext integrity, unforgeability, receiver binding, and sender binding. Confidentiality and ciphertext integrity are the security notions for ciphertexts, while unforgeability, receiver binding, and sender binding are the security notions for reporting tags. Confidentiality guarantees that the information about the messages is not leaked from the ciphertexts and ciphertext integrity guarantees that the ciphertexts cannot be tampered with. Similar to the previous work on message franking [13], we adapt multiple-opening (MO) security for confidentiality and ciphertext integrity. MO security allows to securely encrypt multiple different messages under the same secret key. Unforgeability guarantees that reporting tags are not forged, receiver binding guarantees that the receiver is not able to report messages that were not actually sent, and sender binding guarantees that the sender is not able to send malicious messages that cannot be reported. See Section 3.1 for the details.

We show a construction of a key-evolving message franking scheme combining a CAEAD scheme with a forward secure pseudorandom generator (PRG) and a forward secure message authentication code (MAC) scheme. In a nutshell, we use a CAEAD scheme to generate ciphertexts and franking tags and decrypt them with a secret key updated by a forward secure PRG. Moreover, we use a forward secure MAC scheme to generate and verify reporting tags with an updated tagging key. See Section 4.2 for the details.

### 1.3 Related Work

As mentioned above, Grubbs et al. [13] introduced CAEAD as a core primitive for message franking. They provided two constructions of a CAEAD scheme: Commit-then-Encrypt (CtE) that combines a commitment scheme with an authenticated encryption with associated data (AEAD) scheme and Committing Encrypt-and-PRF (CEP) that uses a nonce-based PRG, a pseudorandom function (PRF), and a collision resistant PRF.

Dodis, Grubbs, Ristenpart, and Woodage [10] showed the attack against message franking for attachments. They also introduced encryptment, which is simplified CAEAD for design and analyse and provided an encryptment scheme using a hash function. Hirose [16] provided an encryptment scheme using a tweakable block cipher (TBC).

Leontiadis and Vaudenay [18] proposed a new security definition, called multiple-opening indistinguishability with partical opening (MO-IND-PO), which ensures confidentiality of unreported parts of the messages after reporting malicious parts. Chen and Tang [6] introduced targeted opening committing authenticated encryption with associated data (TOCE), which allows the receiver to report only the abusive parts of the messages for verification.

Huguenin-Dumittan and Leontiadis [17] introduced message franking channel, which is resistant to replay attacks, out-of-order delivery and message drops.

They provided a construction of a message franking channel using a CAEAD scheme and a MAC scheme.

Tyagi, Grubbs, Len, Miers, and Ristenpart [21] introduced asymmetric message franking (AMF) to achieve content moderation under the condition that the sender and receiver identities are hidden from the service providers. They provided a construction of an AMF scheme using an applied technique of a designated verifier signature scheme.

## 2 Preliminaries

### 2.1 Notation

For a positive integer $n$, we write $[n]$ to denote the set $\{1, \cdots, n\}$. For a finite set $X$, we write $x \xleftarrow{\$} X$ to denote sampling $x$ from $X$ uniformly at random and $|X|$ to denote the cardinality of $X$. For a string $x$, we write $|x|$ to denote the length of $x$. For an algorithm $\mathcal{A}$, we write $y \leftarrow \mathcal{A}(x)$ to denote running $\mathcal{A}$ on the input $x$ to produce the output $y$. $\lambda$ denotes a security parameter. A function $f(\lambda)$ is a negligible function if $f(\lambda)$ tends to 0 faster than $\frac{1}{\lambda^c}$ for every constant $c > 0$. $\mathsf{negl}(\lambda)$ denotes an unspecified negligible function.

### 2.2 Forward Secure Pseudorandom Generator

**Definition 1 (Stateful Pseudorandom Generator [4]).** *A stateful pseudorandom generator* $\mathsf{sPRG} = (\mathsf{Key}, \mathsf{Next})$ *is a tuple of two algorithms associated with a state space* $\mathcal{ST}$ *and a block space* $\mathcal{OUT}$ *defined as follows.*

- $St_0 \leftarrow \mathsf{Key}(1^\lambda, n)$ : *The key generation algorithm* $\mathsf{Key}$ *takes as input a security parameter* $1^\lambda$ *and the total number of time periods* $n$ *and outputs the initial state* $St_0$.
- $(Out_i, St_i) \leftarrow \mathsf{Next}(St_{i-1})$ : *The next step algorithm* $\mathsf{Next}$ *takes as input the current state* $St_{i-1}$ *and outputs a output block* $Out_i$ *and the next state* $St_i$.

**Definition 2 (Forward Security).** *Let* $n \geq 1$ *be any integer. For a stateful pseudorandom generator* $\mathsf{sPRG} = (\mathsf{Key}, \mathsf{Next})$, *we define the forward security game between a challenger* $\mathcal{CH}$ *and an adversary* $\mathcal{A}$ *as follows.*

1. $\mathcal{CH}$ *generates* $St_0 \leftarrow \mathsf{Key}(1^\lambda, n)$, *sets* $i := 0$, *and chooses* $b \xleftarrow{\$} \{0, 1\}$.
2. $\mathcal{CH}$ *sets* $i := i + 1$. *Depending on the value of* $b$, $\mathcal{CH}$ *proceeds as follows.*
   - *If* $b = 1$, $\mathcal{CH}$ *computes* $(Out_i, St_i) \leftarrow \mathsf{Next}(St_{i-1})$ *and sends* $Out_i$ *to* $\mathcal{A}$.
   - *If* $b = 0$, $\mathcal{CH}$ *computes* $(Out'_i, St_i) \leftarrow \mathsf{Next}(St_{i-1})$ *and* $Out_i \xleftarrow{\$} \{0, 1\}^\lambda$ *sends* $Out_i$ *to* $\mathcal{A}$.
3. $\mathcal{A}$ *outputs* $d \in \{0, 1\}$.
4. *If* $d = 1$ *or* $i = n$, $\mathcal{CH}$ *proceeds to the next Step, else repeats Steps 2 and 3.*
5. $\mathcal{CH}$ *sends* $St_i$ *to* $\mathcal{A}$.
6. $\mathcal{A}$ *outputs* $b' \in \{0, 1\}$.

*In this game, we define the advantage of the adversary $\mathcal{A}$ as*

$$\mathsf{Adv}^{\text{FS-PRG}}_{\mathsf{sPRG},\mathcal{A}}(\lambda) := \left|\Pr[b' = b] - \frac{1}{2}\right|.$$

*We say that* sPRG *is forward secure if for any* PPT *adversary $\mathcal{A}$, we have* $\mathsf{Adv}^{\text{FS-PRG}}_{\mathsf{sPRG},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$.

### 2.3 Forward Secure Message Authentication Code

**Definition 3 (Key-Evolving Message Authentication Code [4]).** *A key-evolving message authentication code scheme* $\mathsf{FSMAC} = (\mathsf{Gen}, \mathsf{Upd}, \mathsf{Tag}, \mathsf{Ver})$ *is a tuple of four algorithms associated with a key space $\mathcal{K}$, a message space $\mathcal{M}$, and a tag space $\mathcal{T}$ defined as follows.*

- $\mathsf{K}_0 \leftarrow \mathsf{Gen}(1^\lambda, n)$ : *The key generation algorithm* Gen *takes as input a security parameter $1^\lambda$ and the total number of time period $n$ and outputs the initial secret key $\mathsf{K}_0$.*
- $\mathsf{K}_i \leftarrow \mathsf{Upd}(K_{i-1})$ : *The key update algorithm* Upd *takes as input the current secret key $\mathsf{K}_{i-1}$ and outputs the next secret key $\mathsf{K}_i$.*
- $(\tau, i) \leftarrow \mathsf{Tag}(\mathsf{K}_i, M)$ : *The tagging algorithm* Tag *takes as input the current secret key $\mathsf{K}_i$ and a message $M$ and outputs a tag $\tau$ and the current time period $i$.*
- $b \leftarrow \mathsf{Ver}(\mathsf{K}_i, M, (\tau, \hat{i}))$ : *The verification algorithm* Ver *takes as input the current tagging key $\mathsf{K}_i$, a message $M$, and a pair of a tag and a time period $(\tau, \hat{i})$ and outputs a bit $b$, with 1 meaning accept and 0 meaning reject.*

As the correctness, we require that for any $n, \lambda \in \mathbb{N}$, $M \in \mathcal{M}$, $\mathsf{K}_0 \leftarrow \mathsf{Gen}(1^\lambda, n)$, and $\mathsf{K}_i \leftarrow \mathsf{Upd}(\mathsf{K}_{i-1})$ for $i = 1, \cdots, n$, $\mathsf{Ver}(\mathsf{K}_{\hat{i}}, M, \mathsf{Tag}(\mathsf{K}_{\hat{i}}, M)) = 1$ holds for all $\hat{i} \in [n]$.

**Definition 4 (FS-sEUF-CMA Security).** *Let $n \geq 1$ be some integer. For a key-evolving MAC scheme* FSMAC*, we define the forward secure strong existentially unforgeability under adaptive chosen message attack* (FS-sEUF-CMA *security) game between a challenger $\mathcal{CH}$ and an adversary $\mathcal{A}$ as follows.*

1. *$\mathcal{CH}$ generates $\mathsf{K}_0 \leftarrow \mathsf{Gen}(1^\lambda, n)$ and sets $i := 0$ and $S_t := \emptyset$ for all $t \in [n]$.*
2. *$\mathcal{CH}$ sets $i := i + 1$ and computes $\mathsf{K}_i \leftarrow \mathsf{Upd}(\mathsf{K}_{i-1})$.*
3. *$\mathcal{A}$ is allowed to make tagging queries. On tagging queries $M$, $\mathcal{CH}$ computes $(\tau, i) \leftarrow \mathsf{Tag}(\mathsf{K}_i, M)$, gives $(\tau, i)$ to $\mathcal{A}$, and appends $(M, (\tau, i))$ to $S_i$.*
4. *$\mathcal{A}$ outputs $d \in \{0, 1\}$.*
5. *If $d = 1$ or $i = n$, $\mathcal{CH}$ proceeds to the next Step, else repeats Steps 2 through 4.*
6. *$\mathcal{CH}$ sends $\mathsf{K}_i$ to $\mathcal{A}$.*
7. *$\mathcal{A}$ outputs $(M^*, (\tau^*, i^*))$.*

*In this game, we define the advantage of the adversary $\mathcal{A}$ as*

$$\mathsf{Adv}^{\text{FS-sEUF-CMA}}_{\mathsf{FSMAC},\mathcal{A}}(\lambda) := \Pr[\mathsf{Ver}(\mathsf{K}_{i^*}, M^*, (\tau^*, i^*)) = 1 \wedge (M^*, (\tau^*, i^*)) \notin S_{i^*} \wedge 1 \leq i^* < i].$$

*We say that* FSMAC *is* FS-sEUF-CMA *secure if for any* PPT *adversary $\mathcal{A}$, we have* $\mathsf{Adv}^{\text{FS-sEUF-CMA}}_{\mathsf{FSMAC},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$.

### 2.4 Committing Authenticated Encryption with Associated Data

**Definition 5 (Committing Authenticated Encryption with Associated Data [13]).** *A committing authenticated encryption with associated data (CAEAD) scheme* $\mathsf{CAEAD} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ver})$ *is a tuple of four algorithms associated with a key space* $\mathcal{K}$*, a header space* $\mathcal{H}$*, a message space* $\mathcal{M}$*, an opening space* $\mathcal{O}$*, a ciphertext space* $\mathcal{C}$*, and a franking tag space* $\mathcal{B}$ *defined as follows.*

- $\mathsf{K} \leftarrow \mathsf{Gen}(1^\lambda)$ : *The key generation algorithm* $\mathsf{Gen}$ *takes as input a security parameter* $1^\lambda$ *and outputs a secret key* $\mathsf{K}$.
- $(C_1, C_2) \leftarrow \mathsf{Enc}(\mathsf{K}, H, M)$ : *The encryption algorithm* $\mathsf{Enc}$ *takes as input a secret key* $\mathsf{K}$*, a header* $H$*, and a message* $M$ *and outputs a ciphertext* $C_1$ *and a franking tag* $C_2$.
- $(M', O) \leftarrow \mathsf{Dec}(\mathsf{K}, H, C_1, C_2)$ : *The decryption algorithm* $\mathsf{Dec}$ *takes as input a secret key* $\mathsf{K}$*, a header* $H$*, a ciphertext* $C_1$*, and a franking tag* $C_2$ *and outputs a message* $M'$ *and an opening* $O$.
- $b \leftarrow \mathsf{Ver}(H, M, O, C_2)$ : *The verification algorithm* $\mathsf{Ver}$ *takes as input a header* $H$*, a message* $M$*, an opening* $O$*, and a franking tag* $C_2$ *and outputs a bit* $b$*, with* 1 *meaning accept and* 0 *meaning reject.*

As the correctness, we require that for any $n, \lambda \in \mathbb{N}$, $M \in \mathcal{M}$, $H \in \mathcal{H}$, $\mathsf{K} \leftarrow \mathsf{Gen}(1^\lambda)$, $(C_1, C_2) \leftarrow \mathsf{Enc}(\mathsf{K}, H, M)$, and $(M', O) \leftarrow \mathsf{Dec}(\mathsf{K}, H, C_1, C_2)$, $M' = M$ and $\mathsf{Ver}(H, M', O, C_2) = 1$ holds.

**Definition 6 (MO-IND Security).** *For a CAEAD scheme* $\mathsf{CAEAD}$*, we define the multiple-opening indistinguishability* (MO-IND *security*) *game between a challenger* $\mathcal{CH}$ *and an adversary* $\mathcal{A}$ *as follows.*

1. $\mathcal{CH}$ *generates* $\mathsf{K} \leftarrow \mathsf{Gen}(1^\lambda)$ *and sets* $S := \emptyset$.
2. $\mathcal{A}$ *is allowed to make encryption queries and decryption queries as follows.*
   - *On encryption queries of the form* $(H, M)$*,* $\mathcal{CH}$ *computes* $(C_1, C_2) \leftarrow \mathsf{Enc}(\mathsf{K}, H, M)$*, gives* $(C_1, C_2)$ *to* $\mathcal{A}$*, and appends* $(H, C_1, C_2)$ *to* $S$.
   - *On decryption queries of the form* $(H, C_1, C_2)$*, if* $(H, C_1, C_2) \notin S$*,* $\mathcal{CH}$ *gives* $\perp$ *to* $\mathcal{A}$*. Otherwise,* $\mathcal{CH}$ *computes* $(M', O) \leftarrow \mathsf{Dec}(\mathsf{K}, H, C_1, C_2)$ *and gives* $(M', O)$ *to* $\mathcal{A}$.
3. $\mathcal{A}$ *sends* $(H^*, M_0^*, M_1^*)$ *to* $\mathcal{CH}$*, where* $|M_0^*| = |M_1^*|$.
4. $\mathcal{CH}$ *chooses a challenge bit* $b \xleftarrow{\$} \{0, 1\}$*, computes* $(C_1^*, C_2^*) \leftarrow \mathsf{Enc}(\mathsf{K}, H^*, M_b^*)$ *and sends* $(C_1^*, C_2^*)$ *to* $\mathcal{A}$.
5. $\mathcal{A}$ *is allowed to make the same encryption and decryption queries as with Step 2 except that* $(H^*, C_1^*, C_2^*)$ *cannot be queried in the decryption query.*
6. $\mathcal{A}$ *outputs* $b' \in \{0, 1\}$.

*In this game, we define the advantage of the adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\text{MO-IND}}_{\mathsf{CAEAD},\mathcal{A}}(\lambda) := \left| \Pr[b' = b] - \frac{1}{2} \right|.$$

*We say that* $\mathsf{CAEAD}$ *is* MO-IND *secure if for any* PPT *adversary* $\mathcal{A}$*, we have* $\mathsf{Adv}^{\text{MO-IND}}_{\mathsf{CAEAD},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$.

*Remark 1.* While Grubbs et al. [13] used the definition which guarantees that ciphertexts cannot be distinguished from random strings, our definition ensures that, for the two messages outputed by the adversary, it is hard to identify which message the received ciphertext is generated from. Similar to [6], we can construct a CAEAD scheme satisfying MO-IND security from an authenticated encryption with associated data (AEAD) scheme and a commitment scheme.

**Definition 7 (MO-CTXT Security).** *For a CAEAD scheme* $\mathsf{CAEAD}$*, we define the multiple-opening ciphertext integrity* (MO-CTXT *security*) *game between a challenger* $\mathcal{CH}$ *and an adversary* $\mathcal{A}$ *as follows.*

1. $\mathcal{CH}$ *generates* $\mathsf{K} \leftarrow \mathsf{Gen}(1^\lambda)$ *and sets* $S := \emptyset$.
2. $\mathcal{A}$ *is allowed to make encryption queries and decryption queries as follows.*
   - *On encryption queries of the form* $(H, M)$*,* $\mathcal{CH}$ *computes* $(C_1, C_2) \leftarrow \mathsf{Enc}(\mathsf{K}, H, M)$*, gives* $(C_1, C_2)$ *to* $\mathcal{A}$*, and appends* $(H, C_1, C_2)$ *to* $S$.
   - *On decryption queries of the form* $(H, C_1, C_2)$*,* $\mathcal{CH}$ *computes* $(M', O) \leftarrow \mathsf{Dec}(\mathsf{K}, H, C_1, C_2)$ *and gives* $(M', O)$ *to* $\mathcal{A}$.
3. $\mathcal{A}$ *outputs* $(H^*, C_1^*, C_2^*)$.

*In this game, we define the advantage of the adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\text{MO-CTXT}}_{\mathsf{CAEAD},\mathcal{A}}(\lambda) := \Pr[M^* \neq \perp \wedge (H^*, C_1^*, C_2^*) \notin S : (M^*, O^*) \leftarrow \mathsf{Dec}(\mathsf{K}, H^*, C_1^*, C_2^*)].$$

*We say that* $\mathsf{CAEAD}$ *is* MO-CTXT *secure if for any* PPT *adversary* $\mathcal{A}$*, we have* $\mathsf{Adv}^{\text{MO-CTXT}}_{\mathsf{CAEAD},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$.

**Definition 8 (R-BIND Security).** *We say that a CAEAD scheme* $\mathsf{CAEAD}$ *satisfies the receiver binding* (R-BIND *security*) *if for any* $(H, M) \neq (H', M')$*,* $\mathsf{Ver}(H, M, O, C_2^*) = \mathsf{Ver}(H', M', O', C_2^*) = 1$ *never holds.*

**Definition 9 (S-BIND Security).** *We say that a CAEAD scheme* $\mathsf{CAEAD}$ *satisfies the sender binding* (S-BIND *security*) *if for any* $K \in \mathcal{K}$*,* $H \in \mathcal{H}$*,* $C_1 \in \mathcal{C}$*,* $C_2 \in \mathcal{B}$*, and* $(M', O) \leftarrow \mathsf{Dec}(\mathsf{K}, H, C_1, C_2)$*,* $\mathsf{Ver}(H, M', O, C_2) = 0$ *and* $M' \neq \perp$ *never holds.*

## 3 Forward Secure Message Franking

In this section, we introduce forward secure message franking. First, in Section 3.1, we define the syntax and its correctness of key-evolving message franking. Then, in Section 3.2, we provide forward security definitions for key-evolving message franking.

### 3.1 Syntax

In this section, we provide the syntax of a key-evolving message franking scheme. A key-evolving message franking scheme includes two additional algorithms for updating a secret key and a tagging key asynchronously.

**Definition 10 (Key-Evolving Message Franking).** *A key-evolving message franking scheme* $\mathsf{FSMF}$ *is a tuple of eight algorithms* $(\mathsf{SKGen}, \mathsf{TKGen}, \mathsf{SKUpd}, \mathsf{TKUpd}, \mathsf{Enc}, \mathsf{Tag}, \mathsf{Dec}, \mathsf{Ver})$ *associated with a secret key space* $\mathcal{SK}$*, a tagging key space* $\mathcal{TK}$ *a header space* $\mathcal{H}$*, a message space* $\mathcal{M}$*, an opening space* $\mathcal{O}$*, a ciphertext space* $\mathcal{C}$*, a franking tag space* $\mathcal{B}$*, and a reporting tag space* $\mathcal{T}$ *defined as follow.*

- $\mathsf{SK}_0 \leftarrow \mathsf{SKGen}(1^\lambda, n)$ : *The secret key generation algorithm* $\mathsf{SKGen}$ *takes as input a security parameter* $1^\lambda$ *and the total number of time periods* $n$ *and outputs the initial secret key* $\mathsf{SK}_0$*.*
- $\mathsf{TK}_0 \leftarrow \mathsf{TKGen}(1^\lambda, n)$ : *The tagging key generation algorithm* $\mathsf{TKGen}$ *takes as input a security parameter* $1^\lambda$ *and the total number of time periods* $n$ *and outputs the initial tagging key* $\mathsf{TK}_0$*.*
- $\mathsf{SK}_i \leftarrow \mathsf{SKUpd}(\mathsf{SK}_{i-1})$ : *The secret key update algorithm* $\mathsf{SKUpd}$ *takes as input the current secret key* $\mathsf{SK}_{i-1}$ *and outputs the next secret key* $\mathsf{SK}_i$*.*
- $\mathsf{TK}_j \leftarrow \mathsf{TKUpd}(\mathsf{TK}_{j-1})$ : *The tagging key update algorithm* $\mathsf{TKUpd}$ *takes as input the current tagging key* $\mathsf{TK}_{j-1}$ *and outputs the next tagging key* $\mathsf{TK}_j$*.*
- $(C_1, C_2, i) \leftarrow \mathsf{Enc}(\mathsf{SK}_i, H, M)$ : *The encryption algorithm* $\mathsf{Enc}$ *takes as input the current secret key* $\mathsf{SK}_i$*, a header* $H$*, and a message* $M$ *and outputs a ciphertext* $C_1$*, a franking tag* $C_2$*, and the current time period* $i$*.*
- $(\tau, j) \leftarrow \mathsf{Tag}(\mathsf{TK}_j, C_2)$ : *The tagging algorithm* $\mathsf{Tag}$ *takes as input the current tagging key* $\mathsf{TK}_j$ *and a franking tag* $C_2$ *and outputs a reporting tag* $\tau$ *and the current time period* $j$*.*
- $(M', O) \leftarrow \mathsf{Dec}(\mathsf{SK}_i, H, (C_1, C_2, \hat{i}))$ : *The decryption algorithm* $\mathsf{Dec}$ *takes as input the current secret key* $\mathsf{SK}_i$*, a header* $H$*, and a tuple of a ciphertext, a franking tag, and a time period* $(C_1, C_2, \hat{i})$ *and outputs a message* $M'$ *and an opening* $O$*.*
- $b \leftarrow \mathsf{Ver}(\mathsf{TK}_j, H, M, O, C_2, (\tau, \hat{j}))$ : *The verification algorithm* $\mathsf{Ver}$ *takes as input the current tagging key* $\mathsf{TK}_j$*, a header* $H$*, a message* $M$*, an opening* $O$*, a franking tag* $C_2$*, and a pair of a reporting tag and a time period* $(\tau, \hat{j})$ *and outputs a bit* $b$*, with* 1 *meaning accept and* 0 *meaning reject.*

As the correctness, we require that for any $n, \lambda \in \mathbb{N}$, $M \in \mathcal{M}$, $H \in \mathcal{H}$, $\mathsf{SK}_0 \leftarrow \mathsf{SKGen}(1^\lambda, n)$, $\mathsf{TK}_0 \leftarrow \mathsf{TKGen}(1^\lambda, n)$, $\mathsf{SK}_i \leftarrow \mathsf{SKUpd}(\mathsf{SK}_{i-1})$, $\mathsf{TK}_j \leftarrow \mathsf{TKUpd}(\mathsf{TK}_{j-1})$ for $i, j = 1, \cdots, n$, $(C_1, C_2, \hat{i}) \leftarrow \mathsf{Enc}(\mathsf{SK}_{\hat{i}}, H, M)$, $(\tau, \hat{j}) \leftarrow \mathsf{Tag}(\mathsf{TK}_{\hat{j}}, C_2)$, and $(M', O) \leftarrow \mathsf{Dec}(\mathsf{SK}_{\hat{i}}, H, (C_1, C_2, \hat{i}))$, $M' = M$ and $\mathsf{Ver}(\mathsf{TK}_{\hat{j}}, H, M', O, C_2, (\tau, \hat{j})) = 1$ holds for all $\hat{i}, \hat{j} \in [n]$.

### 3.2 Security Definitions

In this section, we define forward security for five security notions: confidentiality, ciphertext integrity, unforgeability, receiver binding, and sender binding.

*Confidentiality.* Intuitively, confidentiality guarantees that the information of the messages is not leaked from the corresponding ciphertexts. More formally, we require that adversaries with the information of the current secret key cannot distinguish ciphertexts generated by the past secret key. We apply multiple-opening (MO) security [13] to confidentiality. MO security ensures that multiple ciphertexts, encrypted under the same secret key, whose opening is known do not endanger the security of other ciphertexts whose opening is not known. In MO security, the adversaries can make decryption queries in addition to encryption queries to learn the openings of the ciphertexts generated via encryption queries.

**Definition 11 (FS-MO-IND Security).** *Let $n \geq 1$ be some integer. For a key-evolving message franking scheme* $\mathsf{FSMF}$*, we define the forward secure multiple-opening indistinguishability* (FS-MO-IND *security*) *game between a challenger $\mathcal{CH}$ and an adversary $\mathcal{A}$ as follows.*

1. *$\mathcal{CH}$ generates $\mathsf{SK}_0 \leftarrow \mathsf{SKGen}(1^\lambda, n)$ and sets $i := 0$ and $S_t := \emptyset$ for all $t \in [n]$.*
2. *$\mathcal{CH}$ sets $i := i + 1$ and computes $\mathsf{SK}_i \leftarrow \mathsf{SKUpd}(\mathsf{SK}_{i-1})$.*
3. *$\mathcal{A}$ is allowed to make encryption queries and decryption queries as follows.*
   - *On encryption queries of the form $(H, M)$, $\mathcal{CH}$ computes $(C_1, C_2, i) \leftarrow \mathsf{Enc}(\mathsf{SK}_i, H, M)$, gives $(C_1, C_2, i)$ to $\mathcal{A}$, and appends $(H, (C_1, C_2, i))$ to $S_i$.*
   - *On decryption queries of the form $(H, (C_1, C_2, \hat{\imath}))$, if $(H, (C_1, C_2, \hat{\imath})) \notin S_i$, $\mathcal{CH}$ gives $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{CH}$ computes $(M', O) \leftarrow \mathsf{Dec}(\mathsf{SK}_i, H, (C_1, C_2, \hat{\imath}))$ and gives $(M', O)$ to $\mathcal{A}$.*
4. *$\mathcal{A}$ sends $(d, H^*, M_0^*, M_1^*, i^*)$ to $\mathcal{CH}$, where $|M_0^*| = |M_1^*|$.*
5. *If $d = 1$ or $i = n$, $\mathcal{CH}$ proceeds to the next Step, else repeats Steps 2 through 4.*
6. *$\mathcal{CH}$ chooses a challenge bit $b \xleftarrow{\$} \{0, 1\}$. If $i^* \geq i$, $\mathcal{CH}$ chooses $b' \xleftarrow{\$} \{0, 1\}$ and terminates. Otherwise, $\mathcal{CH}$ computes $(C_1^*, C_2^*, i^*) \leftarrow \mathsf{Enc}(\mathsf{SK}_{i^*}, H^*, M_b^*)$ and sends $(\mathsf{SK}_i, (C_1^*, C_2^*, i^*))$ to $\mathcal{A}$.*
7. *$\mathcal{A}$ outputs $b' \in \{0, 1\}$.*

*In this game, we define the advantage of the adversary $\mathcal{A}$ as*

$$\mathsf{Adv}^{\text{FS-MO-IND}}_{\mathsf{FSMF},\mathcal{A}}(\lambda) := \left| \Pr[b' = b] - \frac{1}{2} \right|.$$

*We say that* $\mathsf{FSMF}$ *is* FS-MO-IND *secure if for any* PPT *adversary $\mathcal{A}$, we have* $\mathsf{Adv}^{\text{FS-MO-IND}}_{\mathsf{FSMF},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$.

*Ciphertext Integrity.* Intuitively, ciphertext integrity guarantees that ciphertexts are not tampered with. More formally, we require that adversaries with the information of the current secret key cannot generate a new ciphertext which is correctly decrypted by the past secret key. Similar to the above confidentiality, we apply the MO security to ciphertext integrity.

**Definition 12 (FS-MO-CTXT Security).** *Let $n \geq 1$ be some integer. For a key-evolving message franking scheme* $\mathsf{FSMF}$*, we define the forward secure multiple-opening ciphertext integrity* (FS-MO-CTXT *security*) *game between a challenger $\mathcal{CH}$ and an adversary $\mathcal{A}$ as follows.*

1. $\mathcal{CH}$ generates $\mathsf{SK}_0 \leftarrow \mathsf{SKGen}(1^\lambda, n)$ and sets $i := 0$ and $S_t := \emptyset$ for all $t \in [n]$.
2. $\mathcal{CH}$ sets $i := i+1$ and computes $\mathsf{SK}_i \leftarrow \mathsf{SKUpd}(\mathsf{SK}_{i-1})$.
3. $\mathcal{A}$ is allowed to make encryption queries and decryption queries as follows.
   - On encryption queries of the form $(H, M)$, $\mathcal{CH}$ computes $(C_1, C_2, i) \leftarrow \mathsf{Enc}(\mathsf{SK}_i, H, M)$, gives $(C_1, C_2, i)$ to $\mathcal{A}$, and appends $(H, (C_1, C_2, i))$ to $S_i$.
   - On decryption queries of the form $(H, (C_1, C_2, \hat{i}))$, $\mathcal{CH}$ computes $(M', O) \leftarrow \mathsf{Dec}(\mathsf{SK}_i, H, (C_1, C_2, \hat{i}))$ and gives $(M', O)$ to $\mathcal{A}$.
4. $\mathcal{A}$ outputs $d \in \{0, 1\}$.
5. If $d = 1$ or $i = n$, $\mathcal{CH}$ proceeds to the next Step, else repeats Steps 2 through 4.
6. $\mathcal{CH}$ sends $\mathsf{SK}_i$ to $\mathcal{A}$.
7. $\mathcal{A}$ outputs $(H^*, (C_1^*, C_2^*, i^*))$.

*In this game, we define the advantage of the adversary $\mathcal{A}$ as*

$$\mathsf{Adv}^{\text{FS-MO-CTXT}}_{\mathsf{FSMF},\mathcal{A}}(\lambda) := \Pr[M^* \neq \perp \wedge (H^*, (C_1^*, C_2^*, i^*)) \notin S_{i^*} \wedge 1 \leq i^* < i : (M^*, O^*) \leftarrow \mathsf{Dec}(\mathsf{SK}_{i^*}, H^*, (C_1^*, C_2^*, i^*))].$$

*We say that* $\mathsf{FSMF}$ *is* FS-MO-CTXT *secure if for any* PPT *adversary $\mathcal{A}$, we have* $\mathsf{Adv}^{\text{FS-MO-CTXT}}_{\mathsf{FSMF},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$.

*Unforgeability.* Intuitively, unforgeability guarantees that reporting tags are not forged. More formally, we require that adversaries with the information of the current tagging key cannot generate a new reporting tag which is successfully verified by the past tagging key.

**Definition 13 (FS-UNF Security).** *Let $n \geq 1$ be some integer. For a key-evolving message franking scheme* $\mathsf{FSMF}$*, we define the forward secure unforgeability* (FS-UNF *security*) *game between a challenger $\mathcal{CH}$ and an adversary $\mathcal{A}$ as follows.*

1. $\mathcal{CH}$ generates $\mathsf{TK}_0 \leftarrow \mathsf{TKGen}(1^\lambda, n)$ and sets $j := 0$ and $S_t := \emptyset$ for all $t \in [n]$.
2. $\mathcal{CH}$ sets $j := j+1$ and computes $\mathsf{TK}_j \leftarrow \mathsf{TKUpd}(\mathsf{TK}_{j-1})$.
3. $\mathcal{A}$ is allowed to make tagging queries. On tagging queries $C_2$, $\mathcal{CH}$ computes $(\tau, j) \leftarrow \mathsf{Tag}(\mathsf{TK}_j, C_2)$, gives $(\tau, j)$ to $\mathcal{A}$, and appends $(C_2, (\tau, j))$ to $S_j$.
4. $\mathcal{A}$ outputs $d \in \{0, 1\}$.
5. If $d = 1$ or $j = n$, $\mathcal{CH}$ proceeds to the next Step, else repeats Steps 2 through 4.
6. $\mathcal{CH}$ sends $\mathsf{TK}_j$ to $\mathcal{A}$.
7. $\mathcal{A}$ outputs $(H^*, M^*, O^*, C_2^*, (\tau^*, j^*))$.

*In this game, we define the advantage of the adversary $\mathcal{A}$ as*

$$\mathsf{Adv}^{\text{FS-UNF}}_{\mathsf{FSMF},\mathcal{A}}(\lambda) := \Pr[\mathsf{Ver}(\mathsf{TK}_{j^*}, H^*, M^*, O^*, C_2^*, (\tau^*, j^*)) = 1 \wedge (C_2^*, (\tau^*, j^*)) \notin S_{j^*} \wedge 1 \leq j^* < j].$$

*We say that* $\mathsf{FSMF}$ *is* FS-UNF *secure if for any* PPT *adversary $\mathcal{A}$, we have* $\mathsf{Adv}^{\text{FS-UNF}}_{\mathsf{FSMF},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$.

*Receiver Binding.* Intuitively, receiver binding guarantees that the receiver is not able to report messages which were not actually sent. More formally, we require that adversaries with the information of the current tagging key cannot generate two messages successfully verified by the past tagging key for a pair of franking tag and reporting tag.

**Definition 14 (FS-R-BIND Security).** *Let $n \geq 1$ be some integer. For a key-evolving message franking scheme* $\mathsf{FSMF}$*, we define the forward secure receiver binding* (FS-R-BIND *security*) *game between a challenger $\mathcal{CH}$ and an adversary $\mathcal{A}$ as follows.*

1. *$\mathcal{CH}$ generates $\mathsf{TK}_0 \leftarrow \mathsf{TKGen}(1^\lambda, n)$ and sets $j := 0$.*
2. *$\mathcal{CH}$ sets $j := j + 1$ and computes $\mathsf{TK}_j \leftarrow \mathsf{TKUpd}(\mathsf{TK}_{j-1})$.*
3. *$\mathcal{A}$ is allowed to make tagging queries. On tagging queries of $C_2$, $\mathcal{CH}$ computes $(\tau, j) \leftarrow \mathsf{Tag}(\mathsf{TK}_j, C_2)$ and gives $(\tau, j)$ to $\mathcal{A}$.*
4. *$\mathcal{A}$ outputs $d \in \{0, 1\}$.*
5. *If $d = 1$ or $j = n$, $\mathcal{CH}$ proceeds to the next Step, else repeats Steps 2 through 4.*
6. *$\mathcal{CH}$ sends $\mathsf{TK}_j$ to $\mathcal{A}$.*
7. *$\mathcal{A}$ outputs $((H, M, O), (H', M', O'), C_2^*, (\tau^*, j^*))$.*

*In this game, we define the advantage of the adversary $\mathcal{A}$ as*

$$\mathsf{Adv}_{\mathsf{FSMF},\mathcal{A}}^{\text{FS-R-BIND}}(\lambda) := \Pr[\mathsf{Ver}(\mathsf{TK}_{j^*}, H, M, O, C_2^*, (\tau^*, j^*)) = \mathsf{Ver}(\mathsf{TK}_{j^*}, H', M', O', C_2^*, (\tau^*, j^*)) = 1 \wedge (H, M) \neq (H', M') \wedge 1 \leq j^* < j].$$

*We say that* $\mathsf{FSMF}$ *is* FS-R-BIND *secure if for any* PPT *adversary $\mathcal{A}$, we have* $\mathsf{Adv}_{\mathsf{FSMF},\mathcal{A}}^{\text{FS-R-BIND}}(\lambda) = \mathsf{negl}(\lambda)$.

*Sender Binding.* Intuitively, sender binding guarantees that the sender is not able to send malicious messages that cannot be reported. More formally, we require that adversaries with the information of the current tagging key cannot generate a ciphertext that are correctly decrypted but fail to verify by the past tagging key.

**Definition 15 (FS-S-BIND Security).** *Let $n \geq 1$ be some integer. For a key-evolving message franking scheme* $\mathsf{FSMF}$*, we define the forward secure sender binding* (FS-S-BIND *security*) *game between a challenger $\mathcal{CH}$ and an adversary $\mathcal{A}$ as follows.*

1. *$\mathcal{CH}$ generates $\mathsf{TK}_0 \leftarrow \mathsf{TKGen}(1^\lambda, n)$ and sets $j := 0$.*
2. *$\mathcal{CH}$ sets $j := j + 1$ and computes $\mathsf{TK}_j \leftarrow \mathsf{TKUpd}(\mathsf{TK}_{j-1})$.*
3. *$\mathcal{A}$ is allowed to make tagging queries. On tagging queries of $C_2$, $\mathcal{CH}$ computes $(\tau, j) \leftarrow \mathsf{Tag}(\mathsf{TK}_j, C_2)$ and gives $(\tau, j)$ to $\mathcal{A}$.*
4. *$\mathcal{A}$ outputs $d \in \{0, 1\}$.*
5. *If $d = 1$ or $j = n$, $\mathcal{CH}$ proceeds to the next Step, else repeats Steps 2 through 4.*
6. *$\mathcal{CH}$ sends $\mathsf{TK}_j$ to $\mathcal{A}$.*
7. *$\mathcal{A}$ outputs $(j^*, \mathsf{SK}_i, H, (C_1, C_2, \hat{i}))$.*

| | |
|---|---|
| $\underline{\mathsf{SKGen}(1^\lambda, n)}$<br>$St_0 \leftarrow \mathsf{sPRG.Key}(1^\lambda, n)$<br>Output $\mathsf{SK}_0 = (0, \epsilon, St_0)$ | $\underline{\mathsf{TKGen}(1^\lambda, n)}$<br>$\mathsf{TK}_0 \leftarrow \mathsf{FSMAC.Gen}(1^\lambda, n)$<br>Output $\mathsf{TK}_0$ |
| $\underline{\mathsf{SKUpd}(\mathsf{SK}_{i-1})}$<br>Parse $\mathsf{SK}_{i-1} = (i-1, Out_{i-1}, St_{i-1})$<br>$(Out_i, St_i) \leftarrow \mathsf{sPRG.Next}(St_{i-1})$<br>Output $\mathsf{SK}_i = (i, Out_i, St_i)$ | $\underline{\mathsf{TKUpd}(\mathsf{TK}_{j-1})}$<br>$\mathsf{TK}_j \leftarrow \mathsf{FSMAC.Upd}(\mathsf{TK}_{j-1})$<br>Output $\mathsf{TK}_j$ |
| $\underline{\mathsf{Enc}(\mathsf{SK}_i, H, M)}$<br>Parse $\mathsf{SK}_i = (i, Out_i, St_i)$<br>$(C_1, C_2) \leftarrow \mathsf{CAEAD.Enc}(Out_i, H, M)$<br>Output $(C_1, C_2, i)$ | $\underline{\mathsf{Tag}(\mathsf{TK}_j, C_2)}$<br>$(\tau, j) \leftarrow \mathsf{FSMAC.Tag}(\mathsf{TK}_j, C_2)$<br>Output $(\tau, j)$ |
| $\underline{\mathsf{Dec}(\mathsf{SK}_i, H, (C_1, C_2, \hat{i}))}$<br>Parse $\mathsf{SK}_i = (i, Out_i, St_i)$<br>If $\hat{i} \neq i$, Output $\perp$<br>else $(M', O) \leftarrow \mathsf{CAEAD.Dec}(Out_i, H, C_1, C_2)$<br>Output $(M', O)$ | $\underline{\mathsf{Ver}(\mathsf{TK}_j, H, M, O, C_2, (\tau, \hat{j}))}$<br>If $\mathsf{CAEAD.Ver}(H, M, O, C_2) = 0$<br>Output 0<br>else $b \leftarrow \mathsf{FSMAC.Ver}(\mathsf{TK}_j, C_2, (\tau, \hat{j}))$<br>Output $b$ |

**Fig. 1.** Construction of key-evolving message franking scheme.

*8.* $\mathcal{CH}$ *computes* $(\tau, j^*) \leftarrow \mathsf{Tag}(\mathsf{TK}_{j^*}, C_2)$ *and* $(M', O) \leftarrow \mathsf{Dec}(\mathsf{SK}_i, H, (C_1, C_2, \hat{i}))$*.*

*In this game, we define the advantage of the adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\text{FS-S-BIND}}_{\mathsf{FSMF},\mathcal{A}}(\lambda) := \Pr[\mathsf{Ver}(\mathsf{TK}_{j^*}, H, M', O, C_2, (\tau, j^*)) = 0 \wedge M' \neq \perp \wedge 1 \leq j^* < j].$$

*We say that* FSMF *is* FS-S-BIND *secure if for any* PPT *adversary* $\mathcal{A}$*, we have* $\mathsf{Adv}^{\text{FS-S-BIND}}_{\mathsf{FSMF},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$*.*

# 4 Construction of Key-Evolving Message Franking

In this section, we show our construction of a key-evolving message franking scheme. First, in Section 4.1, we provide the formal description of our construction. Then, in Section 4.2, we give security proofs for our construction.

## 4.1 Construction

Let $\mathsf{sPRG} = (\mathsf{sPRG.Key}, \mathsf{sPRG.Next})$ be a stateful generator, $\mathsf{FSMAC} = (\mathsf{FSMAC.Gen}, \mathsf{FSMAC.Upd}, \mathsf{FSMAC.Tag}, \mathsf{FSMAC.Ver})$ a key-evolving MAC scheme, and $\mathsf{CAEAD} = (\mathsf{CAEAD.Gen}, \mathsf{CAEAD.Enc}, \mathsf{CAEAD.Dec}, \mathsf{CAEAD.Ver})$ a CAEAD scheme. We assume that outputs of sPRG can be used as secret keys of CAEAD. From these, we construct our key-evolving message franking scheme $\mathsf{FSMF} = (\mathsf{SKGen}, \mathsf{TKGen}, \mathsf{SKUpd}, \mathsf{TKUpd}, \mathsf{Enc}, \mathsf{Tag}, \mathsf{Dec}, \mathsf{Ver})$ in Figure 1.

The correctness of the scheme immediately follows from the correctness of CAEAD and FSMAC.

### 4.2 Security Proof

In this section, we show that our construction of $\mathsf{FSMF}$ given in Section 4.1 satisfies security notions defined in Section 3.2.

**Theorem 1 (FS-MO-IND Security).** *If* $\mathsf{sPRG}$ *satisfies forward security and* $\mathsf{CAEAD}$ *satisfies* MO-IND *security, then* $\mathsf{FSMF}$ *satisfies* FS-MO-IND *security.*

*Proof.* Let $\mathcal{A}$ be a PPT adversary that attacks the FS-MO-IND security of $\mathsf{FSMF}$. We introduce the following games $Game_\alpha$ for $\alpha = 0, 1$.

- $Game_0$: $Game_0$ is exactly the same as the game of FS-MO-IND security.
- $Game_1$: $Game_1$ is identical to $Game_0$ except that $\mathcal{CH}$ computes $Out_i \xleftarrow{\$} \{0,1\}^\lambda$ after computing $(Out'_i, St_i) \leftarrow \mathsf{sPRG.Next}(St_{i-1})$.

Let $G_\alpha$ be the event that $\mathcal{A}$ succeeds in guessing the challenge bit in $Game_\alpha$.

**Lemma 1.** *There exists a* PPT *adversary* $\mathcal{B}$ *such that* $|\Pr[G_0] - \Pr[G_1]| = 2 \cdot \mathsf{Adv}^{\text{FS-PRG}}_{\mathsf{sPRG},\mathcal{B}}(\lambda)$.

*Proof.* We construct an adversary $\mathcal{B}$ that attacks the forward security of $\mathsf{sPRG}$, using the adversary $\mathcal{A}$ as follows.

1. $\mathcal{B}$ sets $i := 0$ and $S_t := \emptyset$ for all $t \in [n]$.
2. Upon receiving $Out_i$ from $\mathcal{CH}$, $\mathcal{B}$ sets $i := i + 1$.
3. $\mathcal{B}$ answers encryption queries and decryption queries from $\mathcal{A}$ as follows.
   - On encryption queries of the form $(H, M)$, $\mathcal{B}$ computes $(C_1, C_2) \leftarrow \mathsf{CAEAD.Enc}(Out_i, H, M)$, returns $(C_1, C_2, i)$ to $\mathcal{A}$, and appends $(H, (C_1, C_2, i))$ to $S_i$.
   - On decryption queries of the form $(H, (C_1, C_2, \hat{\imath}))$, if $(H, (C_1, C_2, \hat{\imath})) \notin S_i$, $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{B}$ computes $(M', O) \leftarrow \mathsf{CAEAD.Dec}(Out_i, H, C_1, C_2)$ and returns $(M', O)$ to $\mathcal{A}$.
4. When $\mathcal{A}$ outputs $(d, H^*, M_0^*, M_1^*, i^*)$, $\mathcal{B}$ returns $d$ to $\mathcal{CH}$.
5. $\mathcal{B}$ receives $St_i$ from $\mathcal{CH}$, sets $\mathsf{SK}_i = (i, Out_i, St_i)$, and chooses $g \xleftarrow{\$} \{0,1\}$.
6. If $i^* \geq i$, $\mathcal{B}$ sets $b' := g$, returns $b'$ to $\mathcal{CH}$, and terminates. Otherwise, $\mathcal{B}$ computes $(C_1^*, C_2^*) \leftarrow \mathsf{CAEAD.Enc}(Out_{i^*}, H^*, M_g^*)$ and returns $(\mathsf{SK}_i, (C_1^*, C_2^*, i^*))$ to $\mathcal{A}$.
7. When $\mathcal{A}$ outputs $g'$, if $g' = g$, $\mathcal{B}$ sets $b' := 1$, else $b' := 0$.
8. $\mathcal{B}$ returns $b'$ to $\mathcal{CH}$.

We can see that $\mathcal{B}$ perfectly simulates the game $Game_0$ if $b = 1$ and $Game_1$ if $b = 0$ for $\mathcal{A}$. We assume that $G_\alpha$ occurs. Then, $\mathcal{B}$ outputs $b' = 1$ since $\mathcal{A}$ succeeds in guessing the challenge bit $g$ in $Game_\alpha$. Thus, $\mathsf{Adv}^{\text{FS-PRG}}_{\mathsf{sPRG},\mathcal{B}}(\lambda) = \frac{1}{2} \cdot |\Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0]| = \frac{1}{2} \cdot |\Pr[G_0] - \Pr[G_1]|$ holds. ◻

**Lemma 2.** *There exists a* PPT *adversary* $\mathcal{D}$ *such that* $\left|\Pr[G_1] - \frac{1}{2}\right| = n \cdot \mathsf{Adv}^{\text{MO-IND}}_{\mathsf{CAEAD},\mathcal{D}}(\lambda)$.

*Proof.* We construct an adversary $\mathcal{D}$ that attacks the MO-IND security of $\mathsf{CAEAD}$, using the adversary $\mathcal{A}$ as follows.

1. $\mathcal{D}$ computes $l \leftarrow [n]$ and $St_0 \leftarrow \mathsf{sPRG.Key}(1^\lambda, n)$ and sets $i := 0$ and $S_t := \emptyset$ for all $t \in [n]$.
2. $\mathcal{D}$ sets $i := i+1$, computes $(Out'_i, St_i) \leftarrow \mathsf{sPRG.Next}(St_{i-1})$ and $Out_i \xleftarrow{\$} \{0,1\}^\lambda$ and sets $\mathsf{SK}_i := (i, Out_i, St_i)$.
3. $\mathcal{D}$ answers encryption queries and decryption queries from $\mathcal{A}$ as follows.
   - If $i = l$, on encryption queries of the form $(H, M)$, $\mathcal{D}$ makes encryption queries of the form $(H, M)$ to $\mathcal{CH}$, gets the result $(C_1, C_2)$, returns $(C_1, C_2, i)$ to $\mathcal{A}$, and appends $(H, (C_1, C_2, i))$ to $S_i$.
     On decryption queries of the form $(H, (C_1, C_2, \hat{i}))$, if $(H, (C_1, C_2, \hat{i})) \notin S_i$, $\mathcal{D}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{D}$ makes decryption queries of the form $(H, C_1, C_2)$ to $\mathcal{CH}$, gets the result $(M', O)$, and returns $(M', O)$ to $\mathcal{A}$.
   - If $i \neq l$, on encryption queries of the form $(H, M)$, $\mathcal{D}$ computes $(C_1, C_2) \leftarrow \mathsf{CAEAD.Enc}(Out_i, H, M)$, returns $(C_1, C_2, i)$ to $\mathcal{A}$, and appends $(H, (C_1, C_2, i))$ to $S_i$.
     On decryption queries of the form $(H, (C_1, C_2, \hat{i}))$, if $(H, (C_1, C_2, \hat{i})) \notin S_i$, $\mathcal{D}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{D}$ computes $(M', O) \leftarrow \mathsf{CAEAD.Dec}(Out_i, H, C_1, C_2)$ and returns $(M', O)$ to $\mathcal{A}$.
4. $\mathcal{A}$ outputs $(d, H^*, M_0^*, M_1^*, i^*)$.
5. If $d = 1$ or $i = n$, $\mathcal{D}$ proceeds to the next Step, else repeats Steps 2 through 4.
6. If $i^* \neq l$, $\mathcal{D}$ chooses $b' \xleftarrow{\$} \{0,1\}$, returns $b'$ to $\mathcal{CH}$, and terminates. Otherwise, $\mathcal{D}$ returns $(H^*, M_0^*, M_1^*)$ to $\mathcal{CH}$.
7. $\mathcal{D}$ receives $(C_1^*, C_2^*)$ from $\mathcal{CH}$ and returns $(\mathsf{SK}_i, (C_1^*, C_2^*, i^*))$ to $\mathcal{A}$.
8. When $\mathcal{A}$ outputs $g'$, $\mathcal{D}$ sets $b' := g'$ and returns $b'$ to $\mathcal{CH}$

We can see that $\mathcal{D}$ perfectly simulates the game $Game_1$ for $\mathcal{A}$. We assume that $G_1$ occurs and $i^* = l$ holds or $i^* \neq l$ holds and the challenge bit matches the bit chosen randomly. Then, $\mathcal{D}$ succeeds in guessing the challenge bit in MO-IND security game. Since probability of $i^* = l$ is $\frac{1}{n}$ and probability that the challenge bit matches the bit chosen randomly is $\frac{1}{2}$, $\mathsf{Adv}^{\text{MO-IND}}_{\mathsf{CAEAD},\mathcal{D}}(\lambda) = \left|\left(\frac{1}{n}\Pr[G_1] + \frac{n-1}{n}\cdot\frac{1}{2}\right) - \frac{1}{2}\right| = \frac{1}{n}\cdot\left|\Pr[G_1] - \frac{1}{2}\right|$ holds. □

Combining Lemma 1 and 2, We have

$$\begin{aligned}
\mathsf{Adv}^{\text{FS-MO-IND}}_{\mathsf{FSMF},A}(\lambda) &= \left|\Pr[G_0] - \frac{1}{2}\right| \\
&\leq |\Pr[G_0] - \Pr[G_1]| + \left|\Pr[G_1] - \frac{1}{2}\right| \\
&= 2\cdot\mathsf{Adv}^{\text{FS-PRG}}_{\mathsf{sPRG},\mathcal{B}}(\lambda) + n\cdot\mathsf{Adv}^{\text{MO-IND}}_{\mathsf{CAEAD},\mathcal{D}}(\lambda),
\end{aligned}$$

which concludes the proof of Theorem 1. □

**Theorem 2 (FS-MO-CTXT Security).** *If* $\mathsf{sPRG}$ *satisfies forward security and* $\mathsf{CAEAD}$ *satisfies* MO-CTXT *security, then* $\mathsf{FSMF}$ *satisfies* FS-MO-CTXT *security.*

*Proof.* Let $\mathcal{A}$ be a PPT adversary that attacks the FS-MO-CTXT security of $\mathsf{FSMF}$. We introduce the following games $Game_\alpha$ for $\alpha = 0, 1$.

- $Game_0$: $Game_0$ is exactly the same as the game of FS-MO-CTXT security.
- $Game_1$: $Game_1$ is identical to $Game_0$ except that $\mathcal{CH}$ computes $Out_i \xleftarrow{\$} \{0,1\}^\lambda$ after computing $(Out'_i, St_i) \leftarrow \mathsf{sPRG.Next}(St_{i-1})$.

Let $G_\alpha$ be the event that $\mathcal{A}$ succeeds in outputting the tuple $(H^*, (C_1^*, C_2^*, i^*))$ satisfying

$$M^* \neq \perp, (H^*, (C_1^*, C_2^*, i^*)) \notin S_{i^*}, \text{and } 1 \leq i^* < i$$

in computing $(M^*, O^*) \leftarrow \mathsf{Dec}(\mathsf{SK}_{i^*}, H^*, (C_1^*, C_2^*, i^*))$ in $Game_\alpha$.

**Lemma 3.** *There exists a* PPT *adversary* $\mathcal{B}$ *such that* $|\Pr[G_0] - \Pr[G_1]| = 2 \cdot \mathsf{Adv}^{\text{FS-PRG}}_{\mathsf{sPRG},\mathcal{B}}(\lambda)$.

*Proof.* We construct an adversary $\mathcal{B}$ that attacks the forward security of $\mathsf{sPRG}$, using the adversary $\mathcal{A}$ as follows.

1. $\mathcal{B}$ sets $i := 0$ and $S_t := \emptyset$ for $t = [n]$.
2. Upon receiving $Out_i$ from $\mathcal{CH}$, $\mathcal{B}$ sets $i := i + 1$.
3. $\mathcal{B}$ answers encryption queries and decryption queries from $\mathcal{A}$ as follows.
   - On encryption queries of the form $(H, M)$, $\mathcal{B}$ computes $(C_1, C_2) \leftarrow \mathsf{CAEAD.Enc}(Out_i, H, M)$, returns $(C_1, C_2, i)$ to $\mathcal{A}$, and appends $(H, (C_1, C_2, i))$ to $S_i$.
   - On decryption queries of the form $(H, (C_1, C_2, \hat{i}))$, if $i \neq \hat{i}$, $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{B}$ computes $(M', O) \leftarrow \mathsf{CAEAD.Dec}(Out_i, H, C_1, C_2)$ and returns $(M', O)$ to $\mathcal{A}$.
4. When $\mathcal{A}$ outputs $d$, $\mathcal{B}$ returns $d$ to $\mathcal{CH}$.
5. $\mathcal{B}$ receives $St_i$ from $\mathcal{CH}$, sets $\mathsf{SK}_i := (i, Out_i, St_i)$, and returns $\mathsf{SK}_i$ to $\mathcal{A}$.
6. When $\mathcal{A}$ outputs $(H^*, (C_1^*, C_2^*, i^*))$, $\mathcal{B}$ computes $(M^*, O^*) \leftarrow \mathsf{CAEAD.Dec}(Out_{i^*}, H^*, C_1^*, C_2^*)$. If $M^* \neq \perp$, $(H^*, (C_1^*, C_2^*, i^*)) \notin S_{i^*}$, and $1 \leq i^* < i$, $\mathcal{B}$ sets $b' := 1$, else $b' := 0$.
7. $\mathcal{B}$ returns $b'$ to $\mathcal{CH}$

We can see that $\mathcal{B}$ perfectly simulates the game $Game_0$ if $b = 1$ and $Game_1$ if $b = 0$ for $\mathcal{A}$. We assume that $G_\alpha$ occurs. Then, $\mathcal{B}$ outputs $b' = 1$ since $\mathcal{A}$ succeeds in outputting the tuple $(H^*, (C_1^*, C_2^*, i^*))$ satisfying

$$M^* \neq \perp, (H^*, (C_1^*, C_2^*, i^*)) \notin S_{i^*}, \text{and } 1 \leq i^* < i$$

in computing $(M^*, O^*) \leftarrow \mathsf{Dec}(\mathsf{SK}_{i^*}, H^*, (C_1^*, C_2^*, i^*))$ in $Game_\alpha$. Thus, $\mathsf{Adv}^{\text{FS-PRG}}_{\mathsf{sPRG},\mathcal{B}}(\lambda) = \frac{1}{2} \cdot |\Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]| = \frac{1}{2} \cdot |\Pr[G_0] - \Pr[G_1]|$ holds. ☐

**Lemma 4.** *There exists a* PPT *adversary* $\mathcal{D}$ *such that* $Pr[G_1] = n \cdot \mathsf{Adv}^{\text{MO-CTXT}}_{\mathsf{CAEAD},\mathcal{D}}(\lambda)$.

*Proof.* We construct an adversary $\mathcal{D}$ that attacks the MO-CTXT security of $\mathsf{CAEAD}$, using the adversary $\mathcal{A}$ as follows.

1. $\mathcal{D}$ computes $l \leftarrow [n]$ and $St_0 \leftarrow \mathsf{sPRG.Key}(1^\lambda, n)$ and sets $i := 0$ and $S_t := \emptyset$ for all $t \in [n]$.
2. $\mathcal{D}$ sets $i := i + 1$, computes $(Out'_i, St_i) \leftarrow \mathsf{sPRG.Next}(St_{i-1})$ and $Out_i \leftarrow \{0,1\}^\lambda$, and sets $\mathsf{SK}_i := (i, Out_i, St_i)$.
3. $\mathcal{D}$ answers encryption queries and decryption queries from $\mathcal{A}$ as follows.
   - If $i = l$, on encryption queries of the form $(H, M)$, $\mathcal{D}$ makes encryption queries of the form $(H, M)$ to $\mathcal{CH}$, gets the result $(C_1, C_2)$, returns $(C_1, C_2, i)$ to $\mathcal{A}$, and appends $(H, (C_1, C_2, i))$ to $S_i$.
     On decryption queries of the form $(H, (C_1, C_2, \hat{i}))$, if $i \neq \hat{i}$, $\mathcal{D}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{D}$ makes decryption queries of the form $(H, C_1, C_2)$ to $\mathcal{CH}$, gets the result $(M', O)$, and returns $(M', O)$ to $\mathcal{A}$.
   - If $i \neq l$, on encryption queries of the form $(H, M)$, $\mathcal{D}$ computes $(C_1, C_2) \leftarrow \mathsf{CAEAD.Enc}(Out_i, H, M)$, returns $(C_1, C_2, i)$ to $\mathcal{A}$, and appends $(H, (C_1, C_2, i))$ to $S_i$.
     On decryption queries of the form $(H, (C_1, C_2, \hat{i}))$, if $i \neq \hat{i}$, $\mathcal{D}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{D}$ computes $(M', O) \leftarrow \mathsf{CAEAD.Dec}(Out_i, H, C_1, C_2)$ and returns $(M', O)$ to $\mathcal{A}$.
4. $\mathcal{A}$ outputs $d$.
5. If $d = 1$ or $i = n$, $\mathcal{D}$ proceeds to the next Step, else repeats Steps 2 through 4.
6. If $i \leq l$, $\mathcal{D}$ terminates, else returns $\mathsf{SK}_i$ to $\mathcal{A}$.
7. When $\mathcal{A}$ outputs $(H^*, (C_1^*, C_2^*, i^*))$, if $i^* = l$, $\mathcal{D}$ returns $(H^*, C_1^*, C_2^*)$ to $\mathcal{CH}$, else terminates.

We can see that $\mathcal{D}$ perfectly simulates the game $Game_1$ for $\mathcal{A}$. We assume that $G_1$ occurs and $i^* = l$ holds. Then, $\mathcal{D}$ successfully outputs the tuple $(H^*, C_1^*, C_2^*)$ satisfying

$$M^* \neq \perp \text{ and } (H^*, C_1^*, C_2^*) \notin S$$

in computing $(M^*, O) \leftarrow \mathsf{CAEAD.Dec}(\mathsf{K}, H^*, C_1^*, C_2^*)$ in MO-CTXT security game. Since probability of $i^* = l$ is $\frac{1}{n}$, $\mathsf{Adv}^{\text{MO-CTXT}}_{\mathsf{CAEAD},\mathcal{D}}(\lambda) = \frac{1}{n} \cdot \Pr[G_1]$ holds. □

Combining Lemma 3 and 4, We have

$$\begin{aligned} \mathsf{Adv}^{\text{FS-MO-CTXT}}_{\mathsf{FSMF},A}(\lambda) &= \Pr[G_0] \\ &\leq |\Pr[G_0] - \Pr[G_1]| + \Pr[G_1] \\ &= 2 \cdot \mathsf{Adv}^{\text{FS-PRG}}_{\mathsf{sPRG},\mathcal{B}}(\lambda) + n \cdot \mathsf{Adv}^{\text{MO-CTXT}}_{\mathsf{CAEAD},\mathcal{D}}(\lambda), \end{aligned}$$

which concludes the proof of Theorem 2. □

**Theorem 3 (FS-UNF Security).** *If* FSMAC *satisfies* FS-sEUF-CMA *security, then* FSMF *satisfies* FS-UNF *security.*

*Proof.* Let $\mathcal{A}$ be a PPT adversary that attacks the FS-UNF security of FSMF. We construct an adversary $\mathcal{B}$ that attacks the FS-sEUF-CMA security of FSMAC, using the adversary $\mathcal{A}$ as follows.

1. $\mathcal{B}$ sets $j := 0$ and $T_t := \emptyset$ for all $t \in [n]$.

2. $\mathcal{B}$ sets $j := j + 1$.
3. $\mathcal{B}$ answers tagging queries of the form $C_2$ from $\mathcal{A}$ as follows. $\mathcal{B}$ makes tagging queries of the form $C_2$ to $\mathcal{CH}$, gets the result $(\tau, j)$, returns $(\tau, j)$ to $\mathcal{A}$, and appends $(C_2, (\tau, j))$ to $T_j$.
4. When $\mathcal{A}$ outputs $d$, $\mathcal{B}$ returns $d$ to $\mathcal{CH}$.
5. $\mathcal{B}$ receives $\mathsf{K}_j$ from $\mathcal{CH}$, sets $\mathsf{TK}_j = \mathsf{K}_j$, and returns $\mathsf{TK}_j$ to $\mathcal{A}$.
6. When $\mathcal{A}$ outputs $(H^*, M^*, O^*, C_2^*, (\tau^*, j^*))$, $\mathcal{B}$ returns $(C_2^*, (\tau^*, j^*))$ to $\mathcal{CH}$, else terminates.

We can see that $\mathcal{B}$ perfectly simulates the FS-sEUF-CMA security game for $\mathcal{A}$. We assumes that $\mathcal{A}$ successfully outputs the tuple $(H^*, M^*, O^*, C_2^*, (\tau^*, j^*))$ satisfying

$$\mathsf{Ver}(\mathsf{TK}_{j^*}, H^*, M^*, O^*, C_2^*, (\tau^*, j^*)) = 1, (C_2^*, (\tau^*, j^*)) \notin T_{j^*}, \text{and } 1 \leq j^* < j.$$

Then, $\mathcal{B}$ successfully outputs $(C_2^*, (\tau^*, j^*))$ satisfying

$$\mathsf{FSMAC.Ver}(\mathsf{K}_{j^*}, C_2^*, (\tau^*, j^*)) = 1 \text{ and } (C_2^*, (\tau^*, j^*)) \notin S_{j^*}$$

in FS-sEUF-CMA security game. Thus, $\mathsf{Adv}^{\text{FS-sEUF-CMA}}_{\mathsf{FSMAC},\mathcal{B}}(\lambda) = \mathsf{Adv}^{\text{FS-UNF}}_{\mathsf{FSMF},\mathcal{A}}(\lambda)$, which concludes the proof of Theorem 3. □

**Theorem 4 (FS-R-BIND Security).** *If* CAEAD *satisfies* R-BIND*, then* FSMF *satisfies* FS-R-BIND *security.*

*Proof.* Let $\mathcal{A}$ be a PPT adversary that attacks the FS-R-BIND of FSMF. We assume that $\mathcal{A}$ successfully outputs the tuple $((H, M, O), (H', M', O'), C_2^*, (\tau^*, j^*))$ satisfying

$$\mathsf{Ver}(\mathsf{TK}_{j^*}, H, M, O, C_2^*, (\tau^*, j^*)) = \mathsf{Ver}(\mathsf{TK}_{j^*}, H', M', O', C_2^*, (\tau^*, j^*)) = 1,$$
$$(H, M) \neq (H', M'), \text{and } 1 \leq j^* < j.$$

Then, $\mathsf{CAEAD.Ver}(H, M, O, C_2^*) = \mathsf{CAEAD.Ver}(H', M', O', C_2^*) = 1$ and $(H, M) \neq (H', M')$ holds. This contradicts the R-BIND security of CAEAD. Thus, Theorem 4 holds. □

**Theorem 5 (FS-S-BIND security).** *If* CAEAD *satisfies* S-BIND*, then* FSMF *satisfies* FS-S-BIND *security.*

*Proof.* Let $\mathcal{A}$ be a PPT adversary that attacks the FS-S-BIND of FSMF. We assume that $\mathcal{A}$ successfully outputs the tuple $(j^*, \mathsf{SK}_i, H, C_1, C_2)$ satisfying

$$\mathsf{Ver}(\mathsf{TK}_{j^*}, H, M', O, C_2, (\tau, j^*)) = 0, M' \neq \perp, \text{and } 1 \leq j^* < j$$

in computing $(\tau, j^*) \leftarrow \mathsf{Tag}(\mathsf{TK}_{j^*}, C_2)$ and $(M', O) \leftarrow \mathsf{Dec}(\mathsf{SK}_i, H, C_1, C_2)$.

When $\mathsf{Ver}(\mathsf{TK}_{j^*}, H, M', O, C_2, (\tau, j^*)) = 0$ holds, at least one of $\mathsf{CAEAD.Ver}(H, M', O, C_2) = 0$ and $\mathsf{FSMAC.Ver}(\mathsf{K}_{j^*}, C_2, \tau) = 0$ holds.

If $\mathsf{FSMAC.Ver}(\mathsf{K}_{j^*}, C_2, \tau) = 0$ holds, $\mathsf{FSMAC.Ver}(\mathsf{K}_{j^*}, C_2, \mathsf{FSMAC.Tag}(\mathsf{K}_{j^*}, C_2)) = 0$ holds. This contradicts the correctness of FSMAC.

If $\mathsf{CAEAD.Ver}(H, M', O, C_2) = 0$ holds, $M' \neq \perp$ and $\mathsf{CAEAD.Ver}(H, M, O, C_2) = 0$ holds in computing $(M', O) \leftarrow \mathsf{CAEAD.Dec}(Out_i, H, C_1, C_2)$. This contradicts the S-BIND security of CAEAD. Thus, Theorem 5 holds. □

## 5 Conclusion

In this work, we propose forward secure message franking. Firstly, we formalize key-evolving message franking including additional key update algorithms. Then, we propose forward security for five security requirements. Finally, we show key-evolving message franking satisfying forward security based on committing authenticated encryption with associated data, forward secure pseudorandom generator, and forward secure message authentication code.

Our definition is based on CAEAD introduced by Grubbs et al. [13]. In [6, 18], variants of CAEAD were also proposed. By applying forward secure PRG and forward secure MAC to these schemes as a similar manner to our construction, it seems that forward secure variants schemes are obtained.

**Acknowledgements.** A part of this work was supported by iJST OPERA JPMJOP1612, JST CREST JPMJCR14D6, JPMJCR2113, JSPS KAKENHI JP16H01705, JP17H01695, JP19J22363, JP20J14338, 21H04879.

## References

1. Alwen, J., Coretti, S., Dodis, Y.: The double ratchet: Security notions, proofs, and modularization for the signal protocol. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11476, pp. 129–158. Springer (2019). https://doi.org/10.1007/978-3-030-17653-2_5, https://doi.org/10.1007/978-3-030-17653-2_5
2. Aviram, N., Gellert, K., Jager, T.: Session resumption protocols and efficient forward security for TLS 1.3 0-rtt. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11477, pp. 117–150. Springer (2019). https://doi.org/10.1007/978-3-030-17656-3_5, https://doi.org/10.1007/978-3-030-17656-3_5
3. Bellare, M., Miner, S.K.: A forward-secure digital signature scheme. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 431–448. Springer (1999). https://doi.org/10.1007/3-540-48405-1_28, https://doi.org/10.1007/3-540-48405-1_28
4. Bellare, M., Yee, B.S.: Forward-security in private-key cryptography. In: Joye, M. (ed.) Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13-17, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2612, pp. 1–18. Springer (2003). https://doi.org/10.1007/3-540-36563-X_1, https://doi.org/10.1007/3-540-36563-X_1
5. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. In: Biham, E. (ed.) Advances in Cryptology - EUROCRYPT 2003, International

Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2656, pp. 255–271. Springer (2003). https://doi.org/10.1007/3-540-39200-9_16, https://doi.org/10.1007/3-540-39200-9_16
6. Chen, L., Tang, Q.: People who live in glass houses should not throw stones: Targeted opening message franking schemes. IACR Cryptol. ePrint Arch. **2018**, 994 (2018), https://eprint.iacr.org/2018/994
7. Cohn-Gordon, K., Cremers, C.J.F., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the signal messaging protocol. In: 2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017. pp. 451–466. IEEE (2017). https://doi.org/10.1109/EuroSP.2017.27, https://doi.org/10.1109/EuroSP.2017.27
8. Derler, D., Jager, T., Slamanig, D., Striecks, C.: Bloom filter encryption and applications to efficient forward-secret 0-rtt key exchange. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III. Lecture Notes in Computer Science, vol. 10822, pp. 425–455. Springer (2018). https://doi.org/10.1007/978-3-319-78372-7_14, https://doi.org/10.1007/978-3-319-78372-7_14
9. Diffie, W., van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. Des. Codes Cryptogr. **2**(2), 107–125 (1992). https://doi.org/10.1007/BF00124891, https://doi.org/10.1007/BF00124891
10. Dodis, Y., Grubbs, P., Ristenpart, T., Woodage, J.: Fast message franking: From invisible salamanders to encryptment. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10991, pp. 155–186. Springer (2018). https://doi.org/10.1007/978-3-319-96884-1_6, https://doi.org/10.1007/978-3-319-96884-1_6
11. Facebook: Facebook Messenger app (2016), https://www.messenger.com/
12. Facebook: Messenger Secret Conversations technical whitepaper (2016)
13. Grubbs, P., Lu, J., Ristenpart, T.: Message franking via committing authenticated encryption. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10403, pp. 66–97. Springer (2017). https://doi.org/10.1007/978-3-319-63697-9_3, https://doi.org/10.1007/978-3-319-63697-9_3
14. Günther, C.G.: An identity-based key-exchange protocol. In: Quisquater, J., Vandewalle, J. (eds.) Advances in Cryptology - EUROCRYPT '89, Workshop on the Theory and Application of of Cryptographic Techniques, Houthalen, Belgium, April 10-13, 1989, Proceedings. Lecture Notes in Computer Science, vol. 434, pp. 29–37. Springer (1989). https://doi.org/10.1007/3-540-46885-4_5, https://doi.org/10.1007/3-540-46885-4_5
15. Günther, F., Hale, B., Jager, T., Lauer, S.: 0-rtt key exchange with full forward secrecy. In: Coron, J., Nielsen, J.B. (eds.) Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10212, pp. 519–548 (2017). https://doi.org/10.1007/978-3-319-56617-7_18, https://doi.org/10.1007/978-3-319-56617-7_18

16. Hirose, S.: Compactly committing authenticated encryption using tweakable block cipher. In: Kutylowski, M., Zhang, J., Chen, C. (eds.) Network and System Security - 14th International Conference, NSS 2020, Melbourne, VIC, Australia, November 25-27, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12570, pp. 187–206. Springer (2020). https://doi.org/10.1007/978-3-030-65745-1_11, https://doi.org/10.1007/978-3-030-65745-1_11
17. Huguenin-Dumittan, L., Leontiadis, I.: A message franking channel. IACR Cryptol. ePrint Arch. **2018**, 920 (2018), https://eprint.iacr.org/2018/920
18. Leontiadis, I., Vaudenay, S.: Private message franking with after opening privacy. IACR Cryptol. ePrint Arch. **2018**, 938 (2018), https://eprint.iacr.org/2018/938
19. Open Whisper Systems: Signal (2016), https://signal.org/
20. Pointcheval, D., Sanders, O.: Forward secure non-interactive key exchange. In: Abdalla, M., Prisco, R.D. (eds.) Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8642, pp. 21–39. Springer (2014). https://doi.org/10.1007/978-3-319-10879-7_2, https://doi.org/10.1007/978-3-319-10879-7_2
21. Tyagi, N., Grubbs, P., Len, J., Miers, I., Ristenpart, T.: Asymmetric message franking: Content moderation for metadata-private end-to-end encryption. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11694, pp. 222–250. Springer (2019). https://doi.org/10.1007/978-3-030-26954-8_8, https://doi.org/10.1007/978-3-030-26954-8_8
22. Whatsapp: Whatsapp messenger (2016), https://www.whatsapp.com/