Symmetric Asynchronous Ratcheted Communication with Associated Data

Hailun Yan(✉) and Serge Vaudenay

École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
{hailun.yan,serge.vaudenay}@epfl.ch

Abstract. Following up mass surveillance and privacy issues, modern secure communication protocols now seek strong security, such as forward secrecy and post-compromise security, in the face of state exposures. To address this problem, ratcheting was thereby introduced, widely used in real-world messaging protocols like Signal. However, ratcheting comes with a high cost. Recently, Caforio et al. proposed pragmatic constructions which compose a weakly secure "light" protocol and a strongly secure "heavy" protocol, in order to achieve the so-called ratcheting on demand. The light protocol they proposed has still a high complexity.

In this paper, we prove the security of the lightest possible protocol we could imagine, which essentially encrypts then hashes the secret key. We prove it without any random oracle by introducing a new security notion in the standard model. Our protocol composes well with the generic transformation techniques by Caforio et al. to offer high security and performance at the same time.

1 Introduction

A classic communication model usually assumes that the endpoints are secure while the adversary is on the communication channel. However, protocols in recent messaging applications are secured with end-to-end encryption due to the prevalence of malware and system vulnerabilities. They attempt to enable secure communication services by regularly updating (ratcheting) the encryption key. One notable example of ratcheting is the Signal protocol [14] by Open Whisper Systems with its double-ratchet algorithm. It is also widely used in WhatsApp, Skype and many other secure messaging systems.

The term ratcheting on its own does not mean too much, instead, it is an umbrella term for a number of different security guarantees. One security guarantee which is easiest to provide is *forward secrecy*, which, in case of an exposure, prevents the adversary from decrypting messages exchanged in the past. Typically, it is achieved by deleting old states and generating new ones through one-way functions. Moreover, to prevent the adversary from decrypting messages in the future, some source of randomness is added when updating every state to obtain the so-called *future secrecy*, or *backward secrecy*, or *post-compromise security*, or even *self-healing*. The ratcheting technique is mainly related to how keys are used and updated, rather than how they are obtained. We thereby will

K. Aoki and A. Kanaoka (Eds.): IWSEC 2020, LNCS 12231, pp. 184–204, 2020.
https://doi.org/10.1007/978-3-030-58208-1_11

not be concerned with the method of key distribution, regarding the initial keys as created and distributed in a trusted way.

Besides security, there are many other characteristics of communication systems. In a bidirectional two-party secure communication, participants alternate their role as senders and receivers. The modern instant messaging protocols are substantially *asynchronous*. In other words, for a two-party communication, the messages should be transmitted even though the counterpart is not online. Moreover, the moment when a participant wants to send a message is undefined. The participants can use *random roles* (sender or receiver) arbitrarily.

Previous Work. The ratcheting technique was first deployed in real life protocols, such as the off-the-record (OTR) [5] messaging protocol and the Signal protocol [14]. The Signal protocol especially, gained a lot of interest and adopted by a number of other secure messaging apps. A clean description of Signal was posted by the inventors Marlinspike and Perrin [12]. Cohn-Gordon et al. [7] later gave the first academic analysis of Signal security. Recently, Blazy et al. [4] revisited the Signal protocol. They showed some attacks on the original design and proposed SAID, which reshapes Signal into an identity-based asynchronous messaging protocol with authenticated ratcheting.

The first formal definitions of ratcheting security is given by Bellare et al. [3] at CRYPTO 2017. Following their work, a line of studies about ratcheting protocols have been made with different security levels and primitives [1,6,8–10,13]. Some of these results study secure messaging while others are about key-exchange. These works can be considered as equivalent since secure ratcheted communication can reduce to regularly secure key-exchange. At CRYPTO 2018, Poettering and Rösler [13] designed a protocol with "optimal" security (in the sense that we know no better security so far), but using a random oracle, and heavy algorithms such as hierarchical identity-based encryption (HIBE). Jaeger and Stepanovs [9] proposed a similar protocol with security against compromised random coins: with random coin leakage before usage, which also requires HIBE and a random oracle. Durak and Vaudenay [8] proposed a protocol with slightly lower security but relying on neither HIBE nor random oracle. They also prove that public-key encryption is necessary to construct post-compromise secure messaging. At EUROCRYPT 2019, Jost et al. [10] proposed a protocol with security between optimal security and the security of the Durak-Vaudenay protocol. In the same conference, Alwen et al. [1] proposed two other ratcheting protocols with security against adversarially chosen random coins and immediate decryption. Caforio et al. [6] proposed a generic construction of a messaging protocol offering *on-demand ratcheting*, by composing a strongly secure protocol (to be used infrequently) and a weakly secure one. Their construction further generically strengthen protocols by adding the notion of *security awareness*. Recently, Jost et al. [11] modeled the ratcheting components in a unified and composable framework, allowing for their reuse in a modular fashion. Balli et al. [2] modeled optimally secure ratcheted key exchange (RKE) under randomness manipulation and showed that key-update encryption (which is only constructed from HIBE so far) is necessary and sufficient.

In this paper, we are mostly interested in the work by Caforio et al. [6]. They considered message encryption and adapted Durak-Vaudenay protocol to define asynchronous ratcheted communication with associated data (ARCAD). They designed a weakly secure protocol called liteARCAD which is solely based on symmetric cryptography. It achieves provable forward secrecy and excellent software performances. Moreover, they defined a bidirectional secure communication messaging protocol with hybrid on-demand ratcheting. By integrating two ratcheting protocols with different security levels — a strongly secure protocol (such as Durak-Vaudenay protocol) and a weaker but lighter protocol (such as liteARCAD), the hybrid system allows the sender to select which security level he wants to use. When the ratcheting becomes infrequent, the communication system enjoys satisfactory implementation performance thanks to the high efficiency of liteARCAD. Although already quite efficient, liteARCAD has send/receive complexities which can grow linearly with the number of messages. For instance, a participant who sends n messages without receiving any response accumulates n secret keys in his secret state. When he finally receives a message, he must go through all accumulated keys and clean up his state. Furthermore, the typical number of cryptographic operation per message is still high: sending a message requires one hash and $n + 1$ symmetric encryptions. Receiving a message is similar.

Contribution. In this paper we study the simplest protocol we can imagine: we encrypt a message with a secret key then update the key with a hash function. This guarantees one hash and encryption per message. We call this protocol *Encrypt-then-Hash* (EtH). We introduce a new security notion which relates symmetric encryption with key update by hashing. Essentially, we say that the hash of the encryption key is indistinguishable from random. With this notion, we prove the security of EtH in the standard model. We prove that EtH enforces confidentiality and authentication. We deduce that we can use EtH in the generic constructions by Caforio et al. [6].

Organization. In Section 2, we revisit the preliminary notions from Caforio-Durak-Vaudenay [6], all of which are very relevant to our work. In Section 3, we construct a correct and secure symmetric-cryptography-based ARCAD protocol. Meanwhile, we define a new security notion with respect to one-time authenticated encryption and hash function family. Finally, in Section 4, we formally prove that our scheme is secure.

2 Primitives

In this section, we recall some definitions from Caforio et al. [6]. We mark some of the definitions with the reference [6], indicating that these definitions are unchanged except for possible necessary notation changes. Especially, we slightly adapt the definition of asynchronous ratcheted communication with associated data (ARCAD) for symmetric-cryptography-based ARCAD (SARCAD).

Notations and General Definitions. In the following, we will use these notations. We have two participants Alice (A) and Bob (B). Whenever we talk about either one of the participants, we represent it as P, then $\overline{\mathsf{P}}$ refers to P's counterpart. We have two roles send and rec for sender and receiver respectively. We define $\mathsf{send} = \overline{\mathsf{rec}}$ and $\mathsf{rec} = \overline{\mathsf{send}}$. When participants A and B have exclusive roles (like in unidirectional cases), we call them *sender* S and *receiver* R.

Definition 1 (SARCAD)**.** *A symmetric-cryptography-based asynchronous ratcheted communication with associated data (*SARCAD*) consists of the following PPT algorithms:*

- $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$*: This defines the common public parameters* pp*.*
- $\mathsf{Initall}(1^\lambda, \mathsf{pp}) \rightarrow (\mathsf{st}_\mathsf{A}, \mathsf{st}_\mathsf{B})$*: This returns the initial state of* A *and* B*.*
- $\mathsf{Send}(\mathsf{st}_\mathsf{P}, \mathsf{ad}, \mathsf{pt}) \xrightarrow{\$} (\mathsf{st}'_\mathsf{P}, \mathsf{ct})$*: It takes as input a plaintext* pt *and some associated data* ad *and produces a ciphertext* ct *together with an updated state* st'_P*.*
- $\mathsf{Receive}(\mathsf{st}_\mathsf{P}, \mathsf{ad}, \mathsf{ct}) \xrightarrow{\$} (\mathsf{acc}, \mathsf{st}'_\mathsf{P}, \mathsf{pt})$*: It takes as input a ciphertext* ct *and some associated data* ad *and produces a plaintext* pt *together with an updated state* st'_P *and a flag* acc*.*

We consider bidirectional asynchronous communications. Sending/receiving is then refined by the $\mathsf{RATCH}[\mathsf{P}, \mathsf{role}, \cdot, \cdot]$ call as depicted in Fig. 1. To formally capture the communication status and the trivial cases during the communication, Caforio et al. [6] gave a set of definitions (originally defined in [8] and adapted to ARCAD). We do not want to overload this section by redefining the already existing terminology, so we put these less essential definitions in Appendix A for completeness.

Moreover, we will need a symmetric one-time authenticated encryption (OTAE) scheme in our protocol, which consists of a key space $\mathsf{OTAE}.\mathcal{K}(\lambda)$ and the $\mathsf{OTAE.Enc}$ and $\mathsf{OTAE.Dec}$ algorithms. We will also need a hash function family H consisting of a key space $\mathsf{H}.\mathcal{K}(\lambda)$, a domain $\mathsf{H}.\mathcal{D}(\lambda)$, and the $\mathsf{H.Eval}(\mathsf{hk}, \mathsf{sk})$ algorithm which maps $\mathsf{hk} \in \mathsf{H}.\mathcal{K}(\lambda)$ and $\mathsf{sk} \in \mathsf{H}.\mathcal{D}(\lambda)$ to an element of $\mathsf{H}.\mathcal{D}(\lambda)$. We also put the definitions of the above primitives in Appendix A.

Correctness of SARCAD**.** We say that a symmetric ratcheted communication protocol functions correctly if the receiver performs a right decryption and gets exactly the same plaintext as sent by its counterpart. Correctness implies that participant P has received messages in the same order as those sent by participant $\overline{\mathsf{P}}$.

We formally define the $\mathsf{CORRECTNESS}$ game given in Fig. 5 in Appendix B, in which we initialize two participants. Meanwhile, we define variables $\mathsf{sent}^\mathsf{P}_\mathsf{pt}$ (resp. $\mathsf{received}^\mathsf{P}_\mathsf{pt}$) which keeps a list of messages sent (resp. received) by participant P when running Sent (resp. $\mathsf{Receive}$). For each variable ν such as $\mathsf{sent}^\mathsf{P}_\mathsf{pt}$ or st_P relative to participant P, we denote by $\nu(\mathsf{t})$ the value of ν at time t. The scheduling[1] is defined by a sequence sched of tuples of the form either $(\mathsf{P}, \mathsf{send}, \cdot, \cdot)$ or

[1] Scheduling communication is under the control of the adversary except in the $\mathsf{CORRECTNESS}$ game, in which there is no adversary.

Oracle RATCH (P, "send", ad, pt)

1: $\mathsf{pt}_\mathsf{P} \leftarrow \mathsf{pt}$
2: $\mathsf{ad}_\mathsf{P} \leftarrow \mathsf{ad}$
3: $(\mathsf{st}'_\mathsf{P}, \mathsf{ct}_\mathsf{P}) \leftarrow \mathsf{Send}(\mathsf{st}_\mathsf{P}, \mathsf{ad}_\mathsf{P}, \mathsf{pt}_\mathsf{P})$
4: $\mathsf{st}_\mathsf{P} \leftarrow \mathsf{st}'_\mathsf{P}$
5: append $(\mathsf{ad}_\mathsf{P}, \mathsf{pt}_\mathsf{P})$ to $\mathsf{sent}^\mathsf{P}_\mathsf{pt}$
6: append $(\mathsf{ad}_\mathsf{P}, \mathsf{ct}_\mathsf{P})$ to $\mathsf{sent}^\mathsf{P}_\mathsf{ct}$
7: **return** ct_P

Oracle RATCH (P, "rec", ad, ct)

1: $\mathsf{ct}_\mathsf{P} \leftarrow \mathsf{ct}$
2: $\mathsf{ad}_\mathsf{P} \leftarrow \mathsf{ad}$
3: $(\mathsf{acc}, \mathsf{st}'_\mathsf{P}, \mathsf{pt}_\mathsf{P}) \leftarrow \mathsf{Receive}(\mathsf{st}_\mathsf{P}, \mathsf{ad}_\mathsf{P}, \mathsf{ct}_\mathsf{P})$
4: **if** acc **then**
5: $\quad \mathsf{st}_\mathsf{P} \leftarrow \mathsf{st}'_\mathsf{P}$
6: $\quad$ append $(\mathsf{ad}_\mathsf{P}, \mathsf{pt}_\mathsf{P})$ to $\mathsf{received}^\mathsf{P}_\mathsf{pt}$
7: $\quad$ append $(\mathsf{ad}_\mathsf{P}, \mathsf{ct}_\mathsf{P})$ to $\mathsf{received}^\mathsf{P}_\mathsf{ct}$
8: **return** acc

Oracle $\mathsf{EXP}_\mathsf{st}(\mathsf{P})$

1: **return** st_P

Oracle $\mathsf{EXP}_\mathsf{pt}(\mathsf{P})$

1: **return** pt_P

Oracle CHALLENGE(P, ad, ct)

1: **if** $\mathsf{t}_\mathsf{test} \neq \perp$ **then return** $\perp$
2: **if** $\mathsf{b} = 0$ **then** replace pt by a random string of the same length
3: $\mathsf{ct} \leftarrow \mathsf{RATCH}(\mathsf{P}, \text{"send"}, \mathsf{ad}, \mathsf{pt})$
4: $(\mathsf{t}, \mathsf{P}, \mathsf{ad}, \mathsf{pt}, \mathsf{ct})_\mathsf{test} \leftarrow (\mathsf{time}, \mathsf{P}, \mathsf{ad}, \mathsf{pt}, \mathsf{ct})$
5: **return** ct

Fig. 1. Oracles

$(\mathsf{P}, \mathsf{rec}, \cdot, \cdot)$. In this game, the communication between participants uses a waiting queue for messages in each direction. Each participant has a queue of incoming messages and is pulling them in the order they have been pushed in. Sent messages from P are buffered in the queue of $\overline{\mathsf{P}}$.

Definition 2 (Correctness [6]). *We say that a* SARCAD *protocol is* correct *if for all sequence* sched *of tuples of the form* $(\mathsf{P}, \text{"send"}, \mathsf{ad}, \mathsf{pt})$ *or* $(\mathsf{P}, \text{"rec"}, \mathsf{ad}, \mathsf{ct})$*, the game never returns 1. Namely,*

- *at each stage, for each* P, $\mathsf{received}^\mathsf{P}_\mathsf{pt}$ *is prefix*[2] *of* $\mathsf{sent}^{\overline{\mathsf{P}}}_\mathsf{pt}$*, and*
- *each* $\mathsf{RATCH}(\mathsf{P}, \text{"rec"}, \mathsf{ad}, \mathsf{ct})$ *call returns acc=true.*

Security of SARCAD**.** We define the security of SARCAD with IND-CCA notion resp. FORGE notion, which is captured by using the advantage of an adversary playing the IND-CCA resp. FORGE game.

In addition to the RATCH oracles, the adversary can access several other oracles called EXP_st, EXP_pt and CHALLENGE, see Fig. 1.

- RATCH. This oracle is used to ratchet (either to send or to receive), which is essentially the message exchange procedure.
- EXP_st. This oracle is used to obtain the state of a participant, which implies that the adversary can expose the state of Alice or Bob.
- EXP_pt. This oracle is used to obtain the last received message of a participant.

[2] By saying that $\mathsf{received}^\mathsf{P}_\mathsf{pt}$ is prefix of $\mathsf{sent}^{\overline{\mathsf{P}}}_\mathsf{pt}$, we mean that $\mathsf{sent}^{\overline{\mathsf{P}}}_\mathsf{pt}$ is the concatenation of $\mathsf{received}^\mathsf{P}_\mathsf{pt}$ with a (possible empty) list of $(\mathsf{ad}, \mathsf{pt})$ pairs.

– CHALLENGE. This oracle is used (only in the IND-CCA game) to send either the plaintext or a random string.

Following previous work [6,8], we introduce a *cleanness* predicate when defining the security of a SARCAD scheme. The cleanness predicate identifies and captures all trivial ways of attacking. The adversary is not allowed to make a trivial attack when playing games, as defined by the cleanness predicate $\mathsf{C}_{\mathsf{clean}}$ appearing on Step 6 in the games in Fig. 2. Note that identifying the appropriate cleanness predicate $\mathsf{C}_{\mathsf{clean}}$ is not easy. The difficulty is perhaps to clearly forbid all trivial attacks while allowing efficient protocols. For more details and discussions, we refer our readers to [6,8].

Definition 3 ($\mathsf{C}_{\mathsf{clean}}$-IND-CCA **Security** [6]). *Let* $\mathsf{C}_{\mathsf{clean}}$ *be a cleanness predicate. We consider the* $\mathsf{IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{b},\mathsf{C}_{\mathsf{clean}}}$ *game in Fig. 2. We say that* SARCAD *is* $\mathsf{C}_{\mathsf{clean}}$-IND-CCA*-secure if for any PPT adversary* $\mathcal{A}$*, the advantage*

$$\mathsf{Adv}(\mathcal{A}) = \left| \Pr\left[\mathsf{IND\text{-}CCA}^{\mathcal{A}}_{0,\mathsf{C}_{\mathsf{clean}}}(1^\lambda) \to 1\right] - \Pr\left[\mathsf{IND\text{-}CCA}^{\mathcal{A}}_{1,\mathsf{C}_{\mathsf{clean}}}(1^\lambda) \to 1\right] \right|$$

of $\mathcal{A}$ *in* $\mathsf{IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{b},\mathsf{C}_{\mathsf{clean}}}$ *game is negligible.*

Definition 4 ($\mathsf{C}_{\mathsf{clean}}$-FORGE **Security** [6]). *Let* $\mathsf{C}_{\mathsf{clean}}$ *be a cleanness predicate. Consider* $\mathsf{FORGE}^{\mathcal{A}}_{\mathsf{C}_{\mathsf{clean}}}$ *game in Fig. 2. We say that* SARCAD *is* $\mathsf{C}_{\mathsf{clean}}$-FORGE*-secure if for any PPT adversary* $\mathcal{A}$*, the advantage* $\Pr\left[\mathsf{FORGE}^{\mathcal{A}}_{\mathsf{C}_{\mathsf{clean}}}(1^\lambda) \to 1\right]$ *of* $\mathcal{A}$ *in* $\mathsf{FORGE}^{\mathcal{A}}_{\mathsf{C}_{\mathsf{clean}}}$ *is bounded by* ϵ*.*

Game $\mathsf{IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{b},\mathsf{C}_{\mathsf{clean}}}(1^\lambda)$

1: $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$
2: $\mathsf{Initall}(1^\lambda, \mathsf{pp}) \xrightarrow{\$} (\mathsf{st}_\mathrm{A}, \mathsf{st}_\mathrm{B})$
3: set all sent^*_* and $\mathsf{received}^*_*$ variables to $\emptyset$
4: Set $\mathsf{t}_{\mathsf{test}}$ to $\perp$
5: $\mathsf{b}' \leftarrow \mathcal{A}^{\mathsf{RATCH},\mathsf{EXP}_{\mathsf{st}},\mathsf{EXP}_{\mathsf{pt}},\mathsf{CHALLENGE}}(\mathsf{pp})$
6: **if** $\neg \mathsf{C}_{\mathsf{clean}}$ **then return** $\perp$
7: **return** b'

Game $\mathsf{FORGE}^{\mathcal{A}}_{\mathsf{C}_{\mathsf{clean}}}(1^\lambda)$

1: $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$
2: $\mathsf{Initall}(1^\lambda, \mathsf{pp}) \xrightarrow{\$} (\mathsf{st}_\mathrm{A}, \mathsf{st}_\mathrm{B})$
3: set all sent^*_* and $\mathsf{received}^*_*$ variables to $\emptyset$
4: $(\mathsf{P}, \mathsf{ad}, \mathsf{ct}) \leftarrow \mathcal{A}^{\mathsf{RATCH},\mathsf{EXP}_{\mathsf{st}},\mathsf{EXP}_{\mathsf{pt}}}(\mathsf{pp})$
5: $\mathsf{RATCH}(\mathsf{P}, \text{“rec”}, \mathsf{ad}, \mathsf{ct}) \to \mathsf{acc}$
6: **if** $\neg \mathsf{C}_{\mathsf{clean}}$ **then return** 0
7: **if** $\mathsf{acc} = \mathsf{false}$ **then return** 0
8: **if** $(\mathsf{ad}, \mathsf{ct})$ is not a forgery (Def. 9) for P **then return** 0
9: **return** 1

Fig. 2. IND-CCA, FORGE Games

$\mathsf{Setup}(1^\lambda)$	$\mathsf{Initall}(1^\lambda, \mathsf{hk})$
1: $\mathsf{hk} \xleftarrow{\$} \mathsf{H}.\mathcal{K}(\lambda)$ 2: **return** hk	1: $\mathsf{k}, \mathsf{k}' \xleftarrow{\$} \mathsf{OTAE}.\mathcal{K}(\lambda)$ 2: $\mathsf{st}_A \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{k}, \mathsf{k}')$ 3: $\mathsf{st}_B \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{k}', \mathsf{k})$ 4: **return** $(\mathsf{st}_A, \mathsf{st}_B)$
$\mathsf{Send}(\mathsf{st}, \mathsf{ad}, \mathsf{pt})$	$\mathsf{Receive}(\mathsf{st}, \mathsf{ad}, \mathsf{ct})$
1: parse $\mathsf{st} = (1^\lambda, \mathsf{hk}, \mathsf{sk}, \mathsf{rk})$ 2: $\mathsf{ct} \leftarrow \mathsf{OTAE.Enc}(\mathsf{sk}, \mathsf{ad}, \mathsf{pt})$ 3: $\mathsf{sk}' \leftarrow \mathsf{H.Eval}(\mathsf{hk}, \mathsf{sk})$ 4: $\mathsf{st}' \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{sk}', \mathsf{rk})$ 5: **return** $(\mathsf{st}', \mathsf{ct})$	1: parse $\mathsf{st} = (1^\lambda, \mathsf{hk}, \mathsf{sk}, \mathsf{rk})$ 2: $\mathsf{pt} \leftarrow \mathsf{OTAE.Dec}(\mathsf{rk}, \mathsf{ad}, \mathsf{ct})$ 3: **if** $\mathsf{pt} = \perp$ **then** 4: **return** $(\mathsf{false}, \perp, \perp)$ 5: $\mathsf{rk}' \leftarrow \mathsf{H.Eval}(\mathsf{hk}, \mathsf{rk})$ 6: $\mathsf{st}' \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{sk}, \mathsf{rk}')$ 7: **return** $(\mathsf{true}, \mathsf{st}', \mathsf{pt})$

Fig. 3. EtH: our SARCAD scheme

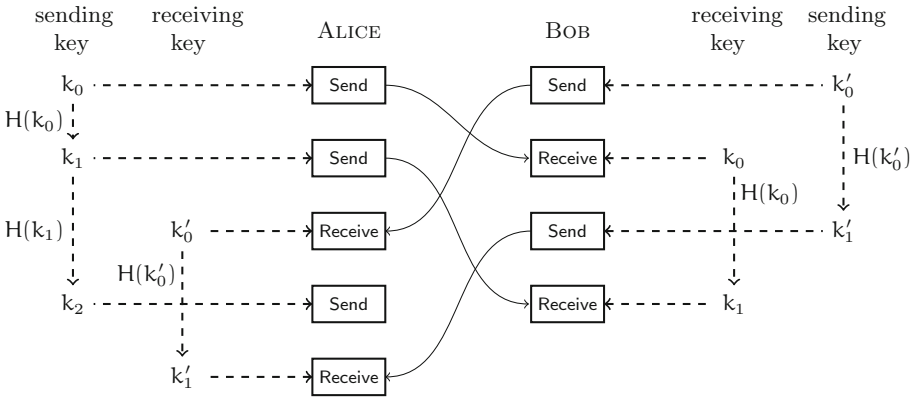

Fig. 4. Message exchanges between Alice and Bob.

3 Construction of a SARCAD Scheme

In this section, we propose a SARCAD scheme, as depicted in Fig. 3. We call it *Encrypt-then-Hash* (EtH), as we encrypt the message with OTAE then update the secret key with H. In this construction, we assume that $\mathsf{OTAE}.\mathcal{K}(\lambda) = \mathsf{H}.\mathcal{D}(\lambda)$. An example of a flow of messages is depicted on Fig. 4. Our scheme guarantees most of the security properties, which can be a perfect alternative to liteARCAD [6]. Note that we do not expect full security (especially the post-compromise security) from this symmetric-only protocol. As pointed by Durak and Vaudenay in [8], a secure and a correct unidirectional ARCAD always implies public-key encryption. We formalize below the security that we achieve for Theorem 2.

In our SARCAD scheme, the participants share two secret symmetric keys as the initial keys, one for encrypting (sending) messages and one for (decrypting) receiving messages. The two communication directions are independent channels in our scheme. Communications are protected by a symmetric one-time authenticated encryption (OTAE) scheme. Moreover, the key states are updated after each communication by a hash function family.

Theorem 1 (Correctness). *Suppose that* OTAE *is correct.* EtH *is a correct* SARCAD*.*

Proof. In the EtH protocol, the two directions of communication are quite independent. Therefore, the correctness of both directions will separately imply the correctness of the whole protocol.

The correctness of the unidirectional case is trivial, which can be easily deduced from the correctness of the OTAE scheme and the deterministic property of the hash function family H. At the beginning, the sender S and the receiver R share the same initial secret key k, and clearly the communication functions correctly at the first step of the loop in the CORRECTNESS game. Suppose that it functions correctly before the i-th step. Let $\mathsf{send}_{\mathsf{pt}}^{\mathsf{S}} = (\mathsf{seq}_1, (\mathsf{ad}, \mathsf{pt}), \mathsf{seq}_2)$ and $\mathsf{received}_{\mathsf{pt}}^{\mathsf{R}} = \mathsf{seq}_1$ at the (i-1)-th step, and let the key used for encrypting/decrypting the last message in seq_1 be k'. Now consider a $\mathsf{RATCH}(\mathsf{R}, \text{“rec”})$ call at the i-th step of the loop, namely $\mathsf{sched}_{\mathsf{i}} = (\mathsf{R}, \text{“rec”}, \mathsf{ad}, \mathsf{ct})$. Let $\mathsf{received}_{\mathsf{pt}}^{\mathsf{R}} = (\mathsf{seq}_1, (\mathsf{ad}', \mathsf{pt}'))$. Note that the key states are updated after each communication by H. Due to the deterministic property of H, the key used by S for encrypting $(\mathsf{ad}, \mathsf{pt})$ (to $(\mathsf{ad}, \mathsf{ct})$) and the key used by R for decrypting $(\mathsf{ad}, \mathsf{ct})$ (to $(\mathsf{ad}', \mathsf{pt}')$) are the same, which is $\mathsf{H}(\mathsf{k}')$. Further, due to the correctness of the OTAE scheme, there must be $(\mathsf{ad}', \mathsf{pt}') = (\mathsf{ad}, \mathsf{pt})$ and $\mathsf{RATCH}(\mathsf{R}, \text{“rec”}, \mathsf{ad}, \mathsf{ct}) = \mathsf{acc}$. □

The SARCAD scheme uses a hash function family with an ad-hoc *pseudorandom* (PR) property.

Definition 5 (PR-Security). *Let* H *be a hash function family and* OTAE *be a one-time authenticated encryption scheme. We say that* H *is pseudorandom for* OTAE *(*PR*) if for any PPT adversary* $\mathcal{A}$ *playing the following game, the advantage*

$$\Pr\left[\mathsf{PR}_0^{\mathcal{A}} \to 1\right] - \Pr\left[\mathsf{PR}_1^{\mathcal{A}} \to 1\right]$$

is negligible.

Game $\mathsf{PR}_{\mathsf{b}}^{\mathcal{A}}$*:*	*Oracle* $\mathsf{ENC}(\mathsf{ad}, \mathsf{pt})$
1: $\mathsf{query}_{\mathsf{enc}} = \bot$	1: ***if*** $\mathsf{query}_{\mathsf{enc}} \neq \bot$ ***then*** *abort*
2: $\mathsf{hk} \xleftarrow{\$} \mathsf{H}.\mathcal{K}(\lambda)$	2: $\mathsf{ct} \leftarrow \mathsf{OTAE.Enc}(\mathsf{k}, \mathsf{ad}, \mathsf{pt})$
3: $\mathsf{k} \xleftarrow{\$} \mathsf{OTAE}.\mathcal{K}(\lambda)$	3: $\mathsf{query}_{\mathsf{enc}} \leftarrow \mathsf{ct}$
4: $\mathsf{k}' \leftarrow \mathsf{H.Eval}(\mathsf{hk}, \mathsf{k})$	4: ***return*** ct
5: ***if*** $\mathsf{b} = 0$ ***then*** *replace* k' *by a random value with the same length*	*Oracle* $\mathsf{DEC}(\mathsf{ad}, \mathsf{ct})$
6: $\mathcal{A}^{\mathsf{ENC},\mathsf{DEC}}(\mathsf{hk}, \mathsf{k}') \to \mathsf{z}$	1: ***return*** $\mathsf{OTAE.Dec}(\mathsf{k}, \mathsf{ad}, \mathsf{ct})$

The PR security allows us to prove forward secrecy without using the random oracle model. In practice, we can use e.g. AES-GCM and SHA-256 truncated to the required length. However, having an extra hashing may be unnecessary as we could build an integrated primitive. We give further discussions on the PR security in a full version of this paper.

When there is a state exposure, it allows simulating every subsequent reception of messages. Therefore, there is no possible healing after a state exposure. To formalize our IND-CCA-security, we prune out post-compromise security but leave forward secrecy by using the following $\mathsf{C}_{\mathsf{sym}}$ predicates from Caforio et al. [6]. Similarly, the notion of trivial forgery changes as the exposure of the state of P now allows to forge for P as well, due to the symmetric key. Thus, a forgery becomes trivial when an $\mathsf{EXP}_{\mathsf{st}}$ occurs. Hence, the FORGE game cannot allow any state exposure at all. We formalize the security by using the $\mathsf{C}_{\mathsf{noexp}}$ cleanness [6] predicate in FORGE-security.

$\mathsf{C}_{\mathsf{noexp}}$: neither A nor B has an $\mathsf{EXP}_{\mathsf{st}}$ before seeing $(\mathsf{ad}_{\mathsf{test}}, \mathsf{ct}_{\mathsf{test}})$.

$\mathsf{C}_{\mathsf{sym}}$: the following conditions are all satisfied:

- there is no $\mathsf{EXP}_{\mathsf{pt}}(\mathsf{P}_{\mathsf{test}})$ after time $\mathsf{t}_{\mathsf{test}}$ until there is a $\mathsf{RATCH}(\mathsf{P}_{\mathsf{test}},.,.,.)$;
- if the CHALLENGE call makes the i-th $\mathsf{RATCH}(\mathsf{P}_{\mathsf{test}},\text{“send”},.,.)$ call and the i-th accepting $\mathsf{RATCH}(\overline{\mathsf{P}}_{\mathsf{test}},\text{“rec”},.,.)$ call occurs in a matching status (Def. 8) at some time $\bar{\mathsf{t}}$, then there is no $\mathsf{EXP}_{\mathsf{pt}}(\overline{\mathsf{P}}_{\mathsf{test}})$ after time $\bar{\mathsf{t}}$ until there is another $\mathsf{RATCH}(\overline{\mathsf{P}}_{\mathsf{test}},.,.,.)$ call;
- ($\mathsf{C}_{\mathsf{noexp}}$) neither A nor B has an $\mathsf{EXP}_{\mathsf{st}}$ before seeing $(\mathsf{ad}_{\mathsf{test}}, \mathsf{ct}_{\mathsf{test}})$.

Theorem 2. *Consider the* SARCAD *scheme* EtH *in Fig. 3. If* OTAE *is* IND-OTCCA*-secure and* SEF-OTCMA*-secure, and* H *is* PR*-secure for* OTAE*, then the scheme is* C_{sym}*-*IND-CCA*-secure and* C_{noexp}*-*FORGE*-secure.*

Proof. We will give the proof in Sect. 4. □

Theorem 1 and Theorem 2 show that EtH satisfies the conditions of the theorems from Caforio et al. [6]. Hence, EtH can play the role of the weakly secure ARCAD in their hybrid “on-demand ratcheting” construction. EtH can also be generically strengthened by the block strengthening [6] to offer security awareness. For instance, if an adversary releases a trivial forgery after a state exposure, the participants can notice it by seeing they can no longer talk with each other. Furthermore, they can see which messages have been received by their counterparts.

4 Security Proof for Our SARCAD Scheme

In this section, we will prove that our scheme is secure (corresponding to Theorem 2).

IND-CCA-Security. We first consider the IND-CCA-security. We define Γ_b the initial $\mathsf{C_{sym}}$-IND-CCA game which has a challenge message $(\mathsf{ad}_{\text{test}}, \mathsf{ct}_{\text{test}})$. We consider the event $\mathsf{C_{noexp}}$ that no participant P has an $\mathsf{EXP_{st}}(\mathsf{P})$ query before having seen $(\mathsf{ad}_{\text{test}}, \mathsf{ct}_{\text{test}})$. The game Γ_b has the property that whenever $\mathsf{C_{sym}}$ does not occur, it never returns 1 due to the $\mathsf{C_{sym}}$ cleanness condition.

We define below for each (Q, m, n) the hybrids $\Gamma^b_{\mathsf{Q},m,n}$ (refer to Fig. 6) which essentially assumes that the challenge message is the n-th message sent by Q. The game maintains two counters: one counter $\mathsf{cnt}^{\mathsf{send}}_{\mathsf{Q}}$ for the number of messages sent by Q and one counter $\mathsf{cnt}^{\mathsf{rec}}_{\overline{\mathsf{Q}}}$ for the number of messages received and accepted by $\overline{\mathsf{Q}}$. For the unidirectional communication from Q to $\overline{\mathsf{Q}}$, the m first session keys are picked randomly, while the following session keys are just updated by hashing the previous one as usual. This is implemented by (1) preparing m randomly-chosen keys $\mathsf{k}_1, \ldots, \mathsf{k}_m$ in the $\mathsf{Initall}$ phase, and (2) modifying the RATCH $(\mathsf{Q}, \text{“send”}, \cdot, \cdot)$ oracle (Line 6–10) and the RATCH $(\overline{\mathsf{Q}}, \text{“rec”}, \cdot, \cdot)$ oracle (Line 8–12). When the challenge message is released, the values of Q and n are verified. If it is incorrect, the game aborts. This is enforced by modifying the $\mathsf{CHALLENGE}$ oracle (Line 5–6). Clearly, we have that

$$\Pr\left[\Gamma_b \to 1\right] = \sum_{\mathsf{Q},n} \Pr\left[\Gamma^b_{\mathsf{Q},1,n} \to 1\right]. \tag{1}$$

In the following, we will prove that for $1 \leqslant m \leqslant n$, the difference between $\Pr\left[\Gamma^b_{\mathsf{Q},m,n} \to 1\right]$ and $\Pr\left[\Gamma^b_{\mathsf{Q},m+1,n} \to 1\right]$ is negligible. Recall that the $(m{+}1)$-th session key k_{m+1} in $\Gamma^b_{\mathsf{Q},m,n}$ is generated by hashing k_m while in $\Gamma^b_{\mathsf{Q},m+1,n}$ it is picked at random. This is the only difference. For any distinguisher $\mathcal{A}$ playing game which is either $\Gamma^b_{\mathsf{Q},m,n}$ or $\Gamma^b_{\mathsf{Q},m+1,n}$, define an adversary $\mathcal{B}$ (refer to Fig. 7) playing game $\mathsf{PR}^{\mathcal{B}}_{b'}$:

Game $\mathsf{PR}^{\mathcal{B}}_{b'}$:

1: $\mathsf{query_{enc}} = \bot$
2: $\mathsf{hk} \xleftarrow{\$} \mathsf{H}.\mathcal{K}(\lambda)$
3: $\mathsf{k}_m \xleftarrow{\$} \mathsf{OTAE}.\mathcal{K}(\lambda)$
4: $\mathsf{k}' \leftarrow \mathsf{H.Eval}(\mathsf{hk}, \mathsf{k}_m)$
5: **if** $b' = 0$ **then** replace k' by a random value with the same length
6: $\mathcal{B}^{\mathsf{ENC},\mathsf{DEC}}(\mathsf{hk}, \mathsf{k}') \to z$

Oracle $\mathsf{ENC}(\mathsf{ad}, \mathsf{pt})$

1: **if** $\mathsf{query_{enc}} \neq \bot$ **then** abort
2: $\mathsf{ct} \leftarrow \mathsf{OTAE.Enc}(\mathsf{k}, \mathsf{ad}, \mathsf{pt})$
3: $\mathsf{query_{enc}} \leftarrow \mathsf{ct}$
4: **return** ct

Oracle $\mathsf{DEC}(\mathsf{ad}, \mathsf{ct})$

1: **return** $\mathsf{OTAE.Dec}(\mathsf{k}, \mathsf{ad}, \mathsf{ct})$

The adversary $\mathcal{B}$ can simulate the difference between $\Gamma^b_{Q,m,n}$ and $\Gamma^b_{Q,m+1,n}$ by using k' he received in game $\mathsf{PR}^{\mathcal{B}}_{b'}$, which is either a hash value or a random value, as the $(m+1)$-th session key k_{m+1}. Finally, $\mathcal{B}$ outputs what $\mathcal{A}$ outputs. We can see that the advantage of $\mathcal{B}$ is

$$\left|\Pr\left[\Gamma^b_{Q,m,n} \to 1\right] - \Pr\left[\Gamma^b_{Q,m+1,n} \to 1\right]\right|,$$

which is negligible due to PR-security (recall Definition. 5). Note that

$$\left|\Pr\left[\Gamma^b_{Q,1,n} \to 1\right] - \Pr\left[\Gamma^b_{Q,n+1,n} \to 1\right]\right|$$

is upper bounded by $\sum_{m=1}^{n}\left|\Pr\left[\Gamma^b_{Q,m,n} \to 1\right] - \Pr\left[\Gamma^b_{Q,m+1,n} \to 1\right]\right|$, thereby also negligible. Combined with Eq. (1), we can then deduce that

$$\left|\Pr\left[\Gamma_b \to 1\right] - \sum_{Q,n}\Pr\left[\Gamma^b_{Q,n+1,n} \to 1\right]\right| \text{ is negligible.} \tag{2}$$

Until now, we have reduced the game $\Gamma^b_{Q,1,n}$ to an ideal case $\Gamma^b_{Q,n+1,n}$. In game $\Gamma^b_{Q,n+1,n}$, $\mathsf{k}_1, \ldots, \mathsf{k}_{n+1}$ are randomly picked. When sending the challenge message, each session key is only used to encrypt/decrypt one message. Moreover, according to $\mathsf{C}_{\mathsf{sym}}$-cleanness, no participant has an $\mathsf{EXP}_{\mathsf{st}}$ before seeing $(\mathsf{ad}_{\text{test}}, \mathsf{ct}_{\text{test}})$, and the plaintext $\mathsf{pt}_{\text{test}}$ corresponding to $(\mathsf{ad}_{\text{test}}, \mathsf{ct}_{\text{test}})$ has no direct or indirect leakage. We can deduce that the difference

$$\left|\Pr\left[\Gamma^0_{Q,n+1,n} \to 1\right] - \Pr\left[\Gamma^1_{Q,n+1,n} \to 1\right]\right|$$

is negligible by the IND-OTCCA security of OTAE. More specifically, for any distinguisher $\mathcal{A}$ playing game which is either $\Gamma^0_{Q,n+1,n}$ or $\Gamma^1_{Q,n+1,n}$, define an adversary $\mathcal{D}$ (refer to Fig. 8) playing game $\mathsf{IND\text{-}OTCCA}^{\mathcal{D}}_b$:

Game $\mathsf{IND\text{-}OTCCA}^{\mathcal{D}}_b$
1: challenge $= \perp$
2: $\mathsf{k}_n \xleftarrow{\$} \mathsf{OTAE}.\mathcal{K}_\lambda$
3: $\mathcal{D}^{\mathsf{CH},\mathsf{DEC}}() \to \mathsf{b}'$
4: **return** b'

Oracle $\mathsf{DEC}(\mathsf{ad}, \mathsf{ct})$
1: **if** $(\mathsf{ad}, \mathsf{ct}) =$ challenge **then** abort
2: **return** $\mathsf{OTAE.Dec}(\mathsf{k}_n, \mathsf{ad}, \mathsf{ct})$

Oracle $\mathsf{CH}(\mathsf{ad}, \mathsf{pt})$
1: **if** challenge $\neq \perp$ **then** abort
2: **if** $\mathsf{b} = 0$ **then**
3: replace pt by a random message of the same length
4: $\mathsf{OTAE.Enc}(\mathsf{k}_n, \mathsf{ad}, \mathsf{pt}) \to \mathsf{ct}$
5: challenge $\leftarrow (\mathsf{ad}, \mathsf{ct})$
6: **return** (ct)

The adversary $\mathcal{D}$ can simulate the difference between $\Gamma^0_{Q,n+1,n}$ and $\Gamma^1_{Q,n+1,n}$ by using the challenge message he received in game $\mathsf{IND\text{-}OTCCA}^{\mathcal{D}}_b$ (where pt is replaced by some random value for $b = 0$) as the challenge message in game $\Gamma^b_{Q,n+1,n}$. Finally, $\mathcal{D}$ outputs what $\mathcal{A}$ outputs. The advantage of $\mathcal{D}$ is

$$\left|\Pr\left[\Gamma^0_{Q,n+1,n} \to 1\right] - \Pr\left[\Gamma^1_{Q,n+1,n} \to 1\right]\right|,$$

which is negligible due to the $\mathsf{IND\text{-}OTCCA}$ security of OTAE.

Finally, we can deduce that the difference $|\Pr[\Gamma_0 \to 1] - \Pr[\Gamma_1 \to 1]|$ is negligible combined with (2).

FORGE-security. We then consider the FORGE-security. We now define Γ as the initial $\mathsf{C_{noexp}\text{-}FORGE}$ game which has a special message $(\mathsf{ad}^*, \mathsf{ct}^*)$. This special message is the forgery sent by the adversary to participant P, where $\mathsf{P} = \overline{\mathsf{Q}}$. The game Γ returns 1 if the special message is a forgery. We also consider the event $\mathsf{C_{noexp}}$ that no participant P has an $\mathsf{EXP_{st}}(\mathsf{P})$ query before having seen $(\mathsf{ad}^*, \mathsf{ct}^*)$. The game Γ has the property that whenever $\mathsf{C_{noexp}}$ does not occur, it never returns 1 due to the $\mathsf{C_{noexp}}$ cleanness condition.

Similarly with the $\mathsf{IND\text{-}CCA}$-security proof, we define below for each (Q, m, n) the hybrids $\Gamma_{Q,m,n}$ (refer to Fig. 9) which essentially assumes that the forgery message is the n-th message sent by Q. We have that

$$\Pr[\Gamma \to 1] = \sum_{Q,n} \Pr[\Gamma_{Q,1,n} \to 1]. \tag{3}$$

Moreover, we have that $|\Pr[\Gamma_{Q,m,n} \to 1] - \Pr[\Gamma_{Q,m+1,n} \to 1]|$ is negligible due to PR-security (Fig. 7 reduction). Therefore,

$$|\Pr[\Gamma_{Q,1,n} \to 1] - \Pr[\Gamma_{Q,n+1,n} \to 1]| \leqslant \sum_{m=1}^{n} |\Pr[\Gamma_{Q,m,n} \to 1] - \Pr[\Gamma_{Q,m+1,n} \to 1]|$$

is negligible. Combined with Eq. (3), we can then deduce that

$$\left|\Pr[\Gamma \to 1] - \sum_{Q,n} \Pr[\Gamma_{Q,n+1,n} \to 1]\right| \text{ is negligible.} \tag{4}$$

The probability that $\Pr[\Gamma_{Q,n+1,n} \to 1]$ is negligible due to the $\mathsf{SEF\text{-}OTCMA}$-security of OTAE. Again, for any distinguisher $\mathcal{A}$ playing game $\Gamma_{Q,n+1,n}$, we define an adversary $\mathcal{E}$ (refer to Fig. 10) playing game $\mathsf{SEF\text{-}OTCMA}^{\mathcal{E}}$:

Game $\mathsf{SEF\text{-}OTCMA}^{\mathcal{E}}$
1: $\mathsf{k}_n \xleftarrow{\$} \mathsf{OTAE}.\mathcal{K}_\lambda$
2: $\mathcal{E}(\lambda) \xrightarrow{\$} \mathsf{sent}^{\mathrm{Q}}_{\mathsf{pt}}[n]$ $(:= \mathsf{ad}, \mathsf{pt})$
3: $\mathsf{OTAE.Enc}(\mathsf{k}_n, \mathsf{ad}, \mathsf{pt}) \to \mathsf{ct}$
4: $\mathcal{E}(\mathsf{ad}, \mathsf{pt}, \mathsf{ct}) \to (\mathsf{ad}^*, \mathsf{ct}^*)$
5: **if** $(\mathsf{ad}^*, \mathsf{ct}^*) = (\mathsf{ad}, \mathsf{ct})$ **then** abort
6: **if** $\mathsf{OTAE.Dec}(\mathsf{k}_n, \mathsf{ad}^*, \mathsf{ct}^*) = \perp$ **then** abort
7: **return** 1

Meanwhile, $\mathcal{E}$ makes a forgery by using the forgery given by $\mathcal{A}$ in game $\Gamma_{\mathrm{Q},n+1,n}$. The advantage of $\mathcal{A}$, which is

$$\Pr\left[\Gamma_{\mathrm{Q},n+1,n} \to 1\right],$$

is upper bounded by the advantage of $\mathcal{E}$, thereby negligible. Finally, we deduce that the probability $\Pr[\Gamma \to 1]$ is negligible combine with (4).

5 Conclusion

We have shown that the simplest Encrypt-then-Hash protocol EtH provides confidentiality and authentication is a very strong sense. Namely, it provides forward secrecy and unforgeability, except for trivial attacks. It can be used as a replacement of liteARCAD in the hybrid ratchet-on-demand protocol with security awareness by Caforio et al. [6]. It provides good complexity advantages. Furthermore, we avoided the use of a random oracle by expliciting a combined security assumption of encryption and hashing (PR security).

A Used Definitions

Function Families. A function family H defines a key space $\mathsf{H}.\mathcal{K}(\lambda)$, a domain $\mathsf{H}.\mathcal{D}(\lambda)$, and a polynomially bounded deterministic algorithm $\mathsf{H.Eval}(\mathsf{hk}, \mathsf{x})$ which takes a key hk in $\mathsf{H}.\mathcal{K}(\lambda)$ and a message x in $\mathsf{H}.\mathcal{D}(\lambda)$ to produce a digest in $\mathsf{H}.\mathcal{D}(\lambda)$.

One-Time Authenticated Encryption (OTAE). An OTAE scheme defines a key space $\mathsf{OTAE}.\mathcal{K}(\lambda)$ and two polynomially bounded deterministic algorithms OTAE.Enc and OTAE.Dec, associated with a message domain $\mathsf{OTAE}.\mathcal{D}(\lambda)$. OTAE.Enc takes a key k in $\mathcal{K}(\lambda)$, an associated data ad and a message pt in $\mathsf{OTAE}.\mathcal{D}(\lambda)$ and returns a string $\mathsf{ct} = \mathsf{OTAE.Enc}(\mathsf{k}, \mathsf{ad}, \mathsf{pt})$. OTAE.Dec takes k, ad and ct and returns a string in $\mathsf{OTAE}.\mathcal{D}(\lambda)$ or else the distinguished symbol $\perp$. It satisfies that

$$\mathsf{OTAE.Dec}(\mathsf{k}, \mathsf{ad}, \mathsf{OTAE.Enc}(\mathsf{k}, \mathsf{ad}, \mathsf{pt})) = \mathsf{pt}$$

for all $\mathsf{k} \in \mathcal{K}(\lambda)$, and $\mathsf{ad}, \mathsf{pt} \in \mathsf{OTAE}.\mathcal{D}(\lambda)$. Moreover, we require that it satisfy the one-time IND-CCA security and SEF-CMA security.

Definition 6 (IND-OTCCA [6]). *An* OTAE *scheme is* IND-OTCCA*-secure, if for any PPT adversary $\mathcal{A}$ playing the following game, the advantage*

$$\Pr\left[\mathsf{IND\text{-}OTCCA}_0^{\mathcal{A}} \to 1\right] - \Pr\left[\mathsf{IND\text{-}OTCCA}_1^{\mathcal{A}} \to 1\right]$$

is negligible.

Game $\mathsf{IND\text{-}OTCCA}_{\mathsf{b}}^{\mathcal{A}}(1^\lambda)$

1: *challenge* $= \bot$
2: $\mathsf{k} \xleftarrow{\$} \mathsf{OTAE}.\mathcal{K}(\lambda)$
3: $\mathcal{A}^{\mathsf{CH},\mathsf{DEC}}(1^\lambda) \to \mathsf{b}'$
4: ***return*** b'

Oracle $\mathsf{DEC}(\mathsf{ad}, \mathsf{ct})$

1: ***if*** $(\mathsf{ad}, \mathsf{ct}) =$ *challenge* ***then*** *abort*
2: ***return*** $\mathsf{OTAE.Dec}(\mathsf{k}, \mathsf{ad}, \mathsf{ct})$

Oracle $\mathsf{CH}(\mathsf{ad}, \mathsf{pt})$

1: ***if*** *challenge* $\neq \bot$ ***then*** *abort*
2: ***if*** $b = 0$ ***then***
3: *replace* pt *by a random message of the same length*
4: $\mathsf{OTAE.Enc}(\mathsf{k}, \mathsf{ad}, \mathsf{pt}) \to \mathsf{ct}$
5: *challenge* $\leftarrow (\mathsf{ad}, \mathsf{ct})$
6: ***return*** ct

Definition 7 (SEF-OTCMA [6]). *An* OTAE *scheme resists to strong existential forgeries under one-time chosen message attacks (*SEF-OTCMA*), if for any PPT adversary $\mathcal{A}$ playing the following game, the advantage* $\Pr\left[\mathsf{SEF\text{-}OTCMA}^{\mathcal{A}} \to 1\right]$ *is negligible.*

Game $\mathsf{SEF\text{-}OTCMA}^{\mathcal{A}}(1^\lambda)$

1: $\mathsf{k} \xleftarrow{\$} \mathsf{OTAE}.\mathcal{K}_\lambda$
2: $\mathcal{A}(1^\lambda) \xrightarrow{\$} (\mathsf{st}, \mathsf{ad}, \mathsf{pt})$
3: $\mathsf{OTAE.Enc}(\mathsf{k}, \mathsf{ad}, \mathsf{pt}) \to \mathsf{ct}$
4: $\mathcal{A}(\mathsf{st}, \mathsf{ad}, \mathsf{pt}, \mathsf{ct}) \to (\mathsf{ad}', \mathsf{ct}')$
5: ***if*** $(\mathsf{ad}', \mathsf{ct}') = (\mathsf{ad}, \mathsf{ct})$ ***then*** *abort*
6: ***if*** $\mathsf{OTAE.Dec}(\mathsf{k}, \mathsf{ad}', \mathsf{ct}') = \bot$ ***then*** *abort*
7: ***return*** *1*

We further append some necessary definitions in ARCAD [6] (adapted from Durak-Vaudenay protocol [8]). For more details, please refer to [6] and [8].

Definition 8 (Matching status [6]). *We say that* P *is in a matching status at time* t *if*

- *at any moment of the game before time* t *for* P, $\mathsf{received}_{\mathsf{ct}}^{\mathsf{P}}$ *is a prefix of* $\mathsf{sent}_{\mathsf{ct}}^{\overline{\mathsf{P}}}$ *— this defines this defines the time* $\overline{\mathsf{t}}$ *for* $\overline{\mathsf{P}}$ *when* $\overline{\mathsf{P}}$ *sent the last message in* $\mathsf{received}_{\mathsf{ct}}^{\mathsf{P}}(\mathsf{t})$.
- *and at any moment of the game before time* $\overline{\mathsf{t}}$ *for* $\overline{\mathsf{P}}$, $\mathsf{received}_{\mathsf{ct}}^{\overline{\mathsf{P}}}$ *is a prefix of* $\mathsf{sent}_{\mathsf{ct}}^{\mathsf{P}}$.

Definition 9 (Forgery [6]). *Given a participant* P *in a game, we say that* $(\mathsf{ad}, \mathsf{ct}) \in \mathsf{received}^{\mathsf{P}}_{\mathsf{ct}}$ *is a forgery if at the moment of the game just before* P *received* $(\mathsf{ad}, \mathsf{ct})$*,* P *was in a matching status, but no longer after receiving* $(\mathsf{ad}, \mathsf{ct})$*.*

B Correctness Game

We formally define the CORRECTNESS game in Fig. 5.

C Hybrids and Adversaries in Security Proof

We define hybrids $\Gamma^{\mathsf{b}}_{\mathrm{Q},\mathrm{m},\mathrm{n}}$ in Fig. 6, adversary $\mathcal{B}$ in Fig. 7, adversary $\mathcal{D}$ in Fig. 8, hybrids $\Gamma_{\mathrm{Q},\mathrm{m},\mathrm{n}}$ in Fig. 9 and adversary $\mathcal{E}$ in Fig. 10.

Game CORRECTNESS(sched)

set all sent^*_* and $\mathsf{received}^*_*$ to $\emptyset$
$\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$
$\mathsf{Initall}(1^\lambda, \mathsf{pp}) \xrightarrow{\$} (\mathsf{st}_\mathsf{A}, \mathsf{st}_\mathsf{B})$
initialize two FIFO lists $\mathsf{incoming}_\mathsf{A}, \mathsf{incoming}_\mathsf{B} \leftarrow \emptyset$
$\mathsf{i} \leftarrow 0$
loop
  $\mathsf{i} \leftarrow \mathsf{i} + 1$
  if $\mathsf{sched}_\mathsf{i}$ of form $(\mathsf{P}, \text{“rec”}, \mathsf{ad}, \mathsf{ct})$ **then**
    if $\mathsf{incoming}_\mathsf{P}$ is empty **then**
      return 0
    pull $(\mathsf{ad}, \mathsf{ct})$ from $\mathsf{incoming}_\mathsf{P}$
    $\mathsf{acc} \leftarrow \mathsf{RATCH}(\mathsf{P}, \text{“rec”}, \mathsf{ad}, \mathsf{ct})$
    if $\mathsf{acc} = \mathsf{false}$ **then return** 1
  else
    parse $\mathsf{sched}_\mathsf{i} = (\mathsf{P}, \text{“send”}, \mathsf{ad}, \mathsf{pt})$
    $\mathsf{ct} \leftarrow \mathsf{RATCH}(\mathsf{P}, \text{“send”}, \mathsf{ad}, \mathsf{pt})$
    push $(\mathsf{ad}, \mathsf{ct})$ to $\mathsf{incoming}_{\overline{\mathsf{P}}}$
  if $\mathsf{received}^{\mathsf{A}}_{\mathsf{pt}}$ not prefix of $\mathsf{sent}^{\mathsf{B}}_{\mathsf{pt}}$ **then return** 1
  if $\mathsf{received}^{\mathsf{B}}_{\mathsf{pt}}$ not prefix of $\mathsf{sent}^{\mathsf{A}}_{\mathsf{pt}}$ **then return** 1

Fig. 5. The CORRECTNESS Game of the SARCAD Protocol

Game $\Gamma^{b}_{Q,m,n}$

1: $hk \xleftarrow{\$} H.\mathcal{K}(\lambda)$
2: $cnt^{send}_{Q}, cnt^{rec}_{\overline{Q}} \leftarrow 0$
3: pick $k_1, \ldots, k_m, k \xleftarrow{\$} \mathcal{K}_\lambda$
4: $st_Q \leftarrow (1^\lambda, hk, k_1, k)$
5: $st_{\overline{Q}} \leftarrow (1^\lambda, hk, k, k_1)$
6: set $sent^*_*$ and $received^*_*$ variables to $\emptyset$
7: Set t_{test} to $\perp$
8: $z \leftarrow \mathcal{A}^{RATCH, EXP_{st}, EXP_{pt}, CHALLENGE}(hk)$
9: **if** $\neg C_{clean}$ **then return** $\perp$
10: **return** z

Oracle RATCH (P, "send", ad, pt)

1: $pt_P \leftarrow pt$
2: $ad_P \leftarrow ad$
3: parse $st_P = (1^\lambda, hk, sk, rk)$
4: $ct_P \leftarrow OTAE.Enc(sk, ad_P, pt_P)$
5: $sk' \leftarrow H.Eval(hk, sk)$
6: **if** $P = Q$ **then**
7: $\quad cnt^{send}_{Q} \leftarrow cnt^{send}_{Q} + 1$
8: $\quad$ **if** $cnt^{send}_{Q} \leqslant m - 1$ **then**
9: $\quad\quad cnt \leftarrow cnt^{send}_{Q} + 1$
10: $\quad\quad sk' \leftarrow k_{cnt}$
11: $st_P \leftarrow (1^\lambda, hk, sk', rk)$
12: append (ad_P, pt_P) to $sent^P_{pt}$
13: append (ad_P, ct_P) to $sent^P_{ct}$
14: **return** ct_P

Oracle RATCH (P, "rec", ad, ct)

1: $ct_P \leftarrow ct$
2: $ad_P \leftarrow ad$
3: parse $st_P = (1^\lambda, hk, sk, rk)$
4: $pt_P \leftarrow OTAE.Dec(rk, ad, ct)$
5: **if** $pt_P = \perp$ **then**
6: $\quad$ **return** false
7: $rk' \leftarrow H.Eval(hk, rk)$
8: **if** $P = \overline{Q}$ **then**
9: $\quad cnt^{rec} \leftarrow cnt^{rec}_{\overline{Q}} + 1$
10: $\quad$ **if** $cnt^{rec}_{\overline{Q}} \leqslant m - 1$ **then**
11: $\quad\quad cnt \leftarrow cnt^{rec}_{\overline{Q}} + 1$
12: $\quad\quad rk' \leftarrow k_{cnt}$
13: $st_P \leftarrow (1^\lambda, hk, sk, rk')$
14: append (ad_P, pt_P) to $received^P_{pt}$
15: append (ad_P, ct_P) to $received^P_{ct}$
16: **return** true

Oracle $EXP_{st}(P)$

1: **if** $(P = Q$ and $cnt^{send}_{Q} \leqslant n)$ or $(P = \overline{Q}$ and $cnt^{rec}_{\overline{Q}} \leqslant n)$ **then** abort
2: **return** st_P

Oracle $EXP_{pt}(P)$

1: **return** pt_P

Oracle CHALLENGE(P, ad, pt)

1: **if** $t_{test} \neq \perp$ **then return** $\perp$
2: **if** $b = 0$ **then**
3: $\quad$ replace pt by a random string of the same length
4: $ct \leftarrow$ RATCH$(P,$ "send", $ad, pt)$
5: **if** $cnt^{send}_{Q} \neq n$ or $P \neq Q$ **then**
6: $\quad$ abort
7: $(t, P, ad, pt, ct)_{test} \leftarrow (time, P, ad, pt, ct)$
8: **return** ct

Fig. 6. IND-CCA-Security: Hybrids $\Gamma^{b}_{Q,m,n}$

$\mathcal{B}^{\mathsf{ENC},\mathsf{DEC}}(\mathsf{hk},\mathsf{k}')$:

1: $\mathsf{hk} \xleftarrow{\$} \mathrm{H}.\mathcal{K}(\lambda)$
2: $\mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}}, \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} \leftarrow 0$
3: pick $\mathsf{k}_1, \ldots, \mathsf{k}_{\mathrm{m}-1}, \mathsf{k} \xleftarrow{\$} \mathcal{K}_\lambda$, $\mathsf{k}_{\mathrm{m}} \leftarrow \perp$
4: $\mathsf{st}_{\mathrm{Q}} \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{k}_1, \mathsf{k})$
5: $\mathsf{st}_{\overline{\mathrm{Q}}} \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{k}, \mathsf{k}_1)$
6: set sent_*^* and $\mathsf{received}_*^*$ variables to $\emptyset$
7: set $\mathsf{t}_{\mathsf{test}}$ to $\perp$
8: $z \leftarrow \mathcal{A}^{\mathsf{RATCH},\mathsf{EXP}_{\mathsf{st}},\mathsf{EXP}_{\mathsf{pt}},\mathsf{CHALLENGE}}(\mathsf{hk})$
9: **if** $\neg \mathsf{C}_{\mathsf{clean}}$ **then return** $\perp$
10: **return** z

Subroutine RATCH (P, "send", ad, pt)

1: $\mathsf{pt}_{\mathrm{P}} \leftarrow \mathsf{pt}$
2: $\mathsf{ad}_{\mathrm{P}} \leftarrow \mathsf{ad}$
3: parse $\mathsf{st}_{\mathrm{P}} = (1^\lambda, \mathsf{hk}, \mathsf{sk}, \mathsf{rk})$
4: **if** $\mathrm{P} = \mathrm{Q}$ and $\mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} = \mathrm{m} - 1$ **then**
5: $\quad \mathsf{ct}_{\mathrm{P}} \leftarrow \mathsf{ENC}(\mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$
6: $\quad \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} = \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} + 1$
7: $\quad \mathsf{sk}' \leftarrow \mathsf{k}'$
8: **else**
9: $\quad \mathsf{ct}_{\mathrm{P}} \leftarrow \mathrm{OTAE.Enc}(\mathsf{sk}, \mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$
10: $\quad \mathsf{sk}' \leftarrow \mathrm{H.Eval}(\mathsf{hk}, \mathsf{sk})$
11: $\quad$ **if** $\mathrm{P} = \mathrm{Q}$ **then**
12: $\quad\quad \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} = \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} + 1$
13: $\quad\quad$ **if** $\mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} \leqslant \mathrm{m} - 1$ **then**
14: $\quad\quad\quad \mathsf{cnt} \leftarrow \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} + 1$
15: $\quad\quad\quad \mathsf{sk}' \leftarrow \mathsf{k}_{\mathsf{cnt}}$
16: $\mathsf{st}_{\mathrm{P}} \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{sk}', \mathsf{rk})$
17: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$ to $\mathsf{sent}_{\mathsf{pt}}^{\mathrm{P}}$
18: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{ct}_{\mathrm{P}})$ to $\mathsf{sent}_{\mathsf{ct}}^{\mathrm{P}}$
19: **return** ct_{P}

Subroutine RATCH (P, "rec", ad, ct)

1: $\mathsf{ct}_{\mathrm{P}} \leftarrow \mathsf{ct}$
2: $\mathsf{ad}_{\mathrm{P}} \leftarrow \mathsf{ad}$
3: parse $\mathsf{st}_{\mathrm{P}} = (1^\lambda, \mathsf{hk}, \mathsf{sk}, \mathsf{rk})$
4: **if** $\mathrm{P} = \overline{\mathrm{Q}}$ and $\mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} = \mathrm{m} - 1$ **then**
5: $\quad \mathsf{pt}_{\mathrm{P}} \leftarrow \mathsf{DEC}(\mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$
6: $\quad$ **if** $\mathsf{pt}_{\mathrm{P}} = \perp$ **then**
7: $\quad\quad$ **return** false
8: $\quad \mathsf{rk}' \leftarrow \mathsf{k}'$
9: $\quad \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} \leftarrow \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} + 1$
10: **else**
11: $\quad \mathsf{pt}_{\mathrm{P}} \leftarrow \mathrm{OTAE.Dec}(\mathsf{rk}, \mathsf{ad}, \mathsf{ct})$
12: $\quad$ **if** $\mathsf{pt}_{\mathrm{P}} = \perp$ **then**
13: $\quad\quad$ **return** false
14: $\quad \mathsf{rk}' \leftarrow \mathrm{H.Eval}(\mathsf{hk}, \mathsf{rk})$
15: $\quad$ **if** $\mathrm{P} = \overline{\mathrm{Q}}$ **then**
16: $\quad\quad \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} \leftarrow \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} + 1$
17: $\quad\quad$ **if** $\mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} \leqslant \mathrm{m} - 1$ **then**
18: $\quad\quad\quad \mathsf{cnt} \leftarrow \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} + 1$
19: $\quad\quad\quad \mathsf{rk}' \leftarrow \mathsf{k}_{\mathsf{cnt}}$
20: $\mathsf{st}_{\mathrm{P}} \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{sk}, \mathsf{rk}')$
21: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$ to $\mathsf{received}_{\mathsf{pt}}^{\mathrm{P}}$
22: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{ct}_{\mathrm{P}})$ to $\mathsf{received}_{\mathsf{ct}}^{\mathrm{P}}$
23: **return** true

Subroutine $\mathsf{EXP}_{\mathsf{st}}(\mathrm{P})$

1: **if** $(\mathrm{P} = \mathrm{Q}$ and $\mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} \leqslant \mathrm{n})$ or $(\mathrm{P} = \overline{\mathrm{Q}}$ and $\mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} \leqslant \mathrm{n})$ **then** abort
2: **return** st_{P}

Subroutine $\mathsf{EXP}_{\mathsf{pt}}(\mathrm{P})$

1: **return** pt_{P}

Subroutine CHALLENGE(P, ad, pt)

1: **if** $\mathsf{t}_{\mathsf{test}} \neq \perp$ **then return** $\perp$
2: **if** $\mathrm{b} = 0$ **then**
3: $\quad$ replace pt by a random string of the same length
4: $\mathsf{ct} \leftarrow \mathsf{RATCH}(\mathrm{P}, \text{"send"}, \mathsf{ad}, \mathsf{pt})$
5: **if** $\mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} \neq \mathrm{n}$ or $\mathrm{P} \neq \mathrm{Q}$ **then**
6: $\quad$ abort
7: $(\mathsf{t}, \mathrm{P}, \mathsf{ad}, \mathsf{pt}, \mathsf{ct})_{\mathsf{test}} \leftarrow (\mathsf{time}, \mathrm{P}, \mathsf{ad}, \mathsf{pt}, \mathsf{ct})$
8: **return** ct

Fig. 7. Adversary $\mathcal{B}$: Simulating $\Gamma_{\mathrm{Q},\mathrm{m},\mathrm{n}}$ and $\Gamma_{\mathrm{Q},\mathrm{m}+1,\mathrm{n}}$

$\mathcal{D}^{\mathsf{DEC},\mathsf{CH}}()$:

1: $\mathsf{hk} \xleftarrow{\$} \mathrm{H}.\mathcal{K}(\lambda)$
2: $\mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}}, \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} \leftarrow 0$
3: pick $\mathsf{k}_1, \ldots, \mathsf{k}_{n-1}, \mathsf{k}_{n+1}, \mathsf{k} \xleftarrow{\$} \mathcal{K}_\lambda$
4: set $\mathsf{k}_n \leftarrow \perp$
5: $\mathsf{st}_{\mathrm{Q}} \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{k}_1, \mathsf{k})$
6: $\mathsf{st}_{\overline{\mathrm{Q}}} \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{k}, \mathsf{k}_1)$
7: set sent_*^* and $\mathsf{received}_*^*$ variables to $\emptyset$
8: set $\mathsf{t}_{\mathsf{test}}$ to $\perp$
9: $\mathrm{b}' \leftarrow \mathcal{A}^{\mathsf{RATCH},\mathsf{EXP}_{\mathsf{st}},\mathsf{EXP}_{\mathsf{pt}},\mathsf{CHALLENGE}}(\mathsf{hk})$
10: **if** $\neg \mathrm{C}_{\mathsf{clean}}$ **then return** $\perp$
11: **return** b'

Subroutine RATCH (P, "send", ad, pt)

1: $\mathsf{pt}_{\mathrm{P}} \leftarrow \mathsf{pt}$
2: $\mathsf{ad}_{\mathrm{P}} \leftarrow \mathsf{ad}$
3: parse $\mathsf{st}_{\mathrm{P}} = (1^\lambda, \mathsf{hk}, \mathsf{sk}, \mathsf{rk})$
4: $\mathsf{ct}_{\mathrm{P}} \leftarrow \mathsf{OTAE.Enc}(\mathsf{sk}, \mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$
5: $\mathsf{sk}' \leftarrow \mathrm{H.Eval}(\mathsf{hk}, \mathsf{sk})$
6: **if** $\mathrm{P} = \mathrm{Q}$ and $\mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} = n$ **then**
7: $\quad \mathsf{ct}_{\mathrm{P}} \leftarrow \mathsf{CH}(\mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$
8: $\quad \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} \leftarrow \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} + 1$
9: $\quad \mathsf{sk}' \leftarrow \mathsf{k}_{n+1}$
10: **else if** $\mathrm{P} = \mathrm{Q}$ **then**
11: $\quad \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} \leftarrow \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} + 1$
12: $\quad$ **if** $\mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} \leqslant n - 1$ **then**
13: $\quad\quad \mathsf{cnt} \leftarrow \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} + 1$
14: $\quad\quad \mathsf{sk}' \leftarrow \mathsf{k}_{\mathsf{cnt}}$
15: $\mathsf{st}_{\mathrm{P}} \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{sk}', \mathsf{rk})$
16: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$ to $\mathsf{sent}_{\mathsf{pt}}^{\mathrm{P}}$
17: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{ct}_{\mathrm{P}})$ to $\mathsf{sent}_{\mathsf{ct}}^{\mathrm{P}}$
18: **return** ct_{P}

Subroutine RATCH (P, "rec", ad, ct)

1: $\mathsf{ct}_{\mathrm{P}} \leftarrow \mathsf{ct}$
2: $\mathsf{ad}_{\mathrm{P}} \leftarrow \mathsf{ad}$
3: parse $\mathsf{st}_{\mathrm{P}} = (1^\lambda, \mathsf{hk}, \mathsf{sk}, \mathsf{rk})$
4: **if** $\mathrm{P} = \overline{\mathrm{Q}}$ and $\mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} = n - 1$ **then**
5: $\quad$ **if** $\mathsf{ad} = \mathsf{ad}_{\mathrm{test}}$ and $\mathsf{ct} = \mathsf{ct}_{\mathrm{test}}$ **then**
6: $\quad\quad \mathsf{pt}_{\mathrm{P}} \leftarrow \mathsf{pt}_{\mathrm{test}}$
7: $\quad$ **else**
8: $\quad\quad \mathsf{pt}_{\mathrm{P}} \leftarrow \mathsf{DEC}(\mathsf{ad}, \mathsf{ct})$
9: $\quad$ **if** $\mathsf{pt}_{\mathrm{P}} = \perp$ **then**
10: $\quad\quad$ **return** false
11: **else**
12: $\quad \mathsf{pt}_{\mathrm{P}} \leftarrow \mathrm{OTAE.Dec}(\mathsf{rk}, \mathsf{ad}, \mathsf{ct})$
13: $\quad$ **if** $\mathsf{pt}_{\mathrm{P}} = \perp$ **then**
14: $\quad\quad$ **return** false
15: $\quad \mathsf{rk}' \leftarrow \mathrm{H.Eval}(\mathsf{hk}, \mathsf{rk})$
16: $\quad$ **if** $\mathrm{P} = \overline{\mathrm{Q}}$ **then**
17: $\quad\quad \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} \leftarrow \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} + 1$
18: $\quad\quad$ **if** $\mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} \leqslant n - 1$ **then**
19: $\quad\quad\quad \mathsf{cnt} \leftarrow \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} + 1$
20: $\quad\quad\quad \mathsf{rk}' \leftarrow \mathsf{k}_{\mathsf{cnt}}$
21: $\mathsf{st}_{\mathrm{P}} \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{sk}, \mathsf{rk}')$
22: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$ to $\mathsf{received}_{\mathsf{pt}}^{\mathrm{P}}$
23: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{ct}_{\mathrm{P}})$ to $\mathsf{received}_{\mathsf{ct}}^{\mathrm{P}}$
24: **return** true

Subroutine $\mathsf{EXP}_{\mathsf{st}}(\mathrm{P})$

1: **if** $(\mathrm{P} = \mathrm{Q}$ and $\mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} \leqslant n)$ or $(\mathrm{P} = \overline{\mathrm{Q}}$ and $\mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} \leqslant n)$ **then** abort
2: **return** st_{P}

Subroutine $\mathsf{EXP}_{\mathsf{pt}}(\mathrm{P})$

1: **return** pt_{P}

Subroutine CHALLENGE(P, ad, pt)

1: **if** $\mathsf{t}_{\mathsf{test}} \neq \perp$ **then return** $\perp$
2: $\mathsf{ct} \leftarrow \mathsf{RATCH}(\mathrm{P}, \text{"send"}, \mathsf{ad}, \mathsf{pt})$
3: **if** $\mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} \neq n$ or $\mathrm{P} \neq \mathrm{Q}$ **then** abort
4: $(\mathsf{t}, \mathrm{P}, \mathsf{ad}, \mathsf{pt}, \mathsf{ct})_{\mathsf{test}} \leftarrow (\mathsf{time}, \mathrm{P}, \mathsf{ad}, \mathsf{pt}, \mathsf{ct})$
5: **return** ct

Fig. 8. Adversary $\mathcal{D}$: Simulating $\Gamma^0_{\mathrm{Q},n+1,n}$ and $\Gamma^1_{\mathrm{Q},n+1,n}$

Game $\Gamma_{Q,m,n}$

1: $\mathsf{hk} \xleftarrow{\$} \mathrm{H}.\mathcal{K}(\lambda)$
2: $\mathsf{cnt}_Q^{\mathsf{send}}, \mathsf{cnt}_{\overline{Q}}^{\mathsf{rec}} \leftarrow 0$
3: pick $\mathsf{k}_1, \ldots, \mathsf{k}_m, \mathsf{k} \xleftarrow{\$} \mathcal{K}_\lambda$
4: $\mathsf{st}_Q \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{k}_1, \mathsf{k})$
5: $\mathsf{st}_{\overline{Q}} \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{k}, \mathsf{k}_1)$
6: set all sent_*^* and $\mathsf{received}_*^*$ variables to $\emptyset$
7: $(\mathsf{P}, \mathsf{ad}^*, \mathsf{ct}^*) \leftarrow \mathcal{A}^{\mathsf{RATCH},\mathsf{EXP}_{\mathsf{st}},\mathsf{EXP}_{\mathsf{pt}}}(\mathsf{hk})$
8: **if** $\mathsf{P} \neq \overline{Q}$ or $\mathsf{cnt}_Q^{\mathsf{send}} \neq n$ **then** abort
9: $\mathsf{RATCH}(\mathsf{P}, \text{“rec”}, \mathsf{ad}^*, \mathsf{ct}^*) \to \mathsf{acc}$
10: **if** $\neg \mathsf{C}_{\mathsf{clean}}$ **then return** 0
11: **if** $\mathsf{acc} = \mathsf{false}$ **then return** 0
12: **if** $(\mathsf{ad}^*, \mathsf{ct}^*)$ is not a forgery for P **then return** 0
13: **return** 1

Oracle RATCH (P, “send”, ad, pt)

1: $\mathsf{pt}_\mathsf{P} \leftarrow \mathsf{pt}$
2: $\mathsf{ad}_\mathsf{P} \leftarrow \mathsf{ad}$
3: parse $\mathsf{st}_\mathsf{P} = (1^\lambda, \mathsf{hk}, \mathsf{sk}, \mathsf{rk})$
4: $\mathsf{ct}_\mathsf{P} \leftarrow \mathrm{OTAE.Enc}(\mathsf{sk}, \mathsf{ad}_\mathsf{P}, \mathsf{pt}_\mathsf{P})$
5: $\mathsf{sk}' \leftarrow \mathrm{H.Eval}(\mathsf{hk}, \mathsf{sk})$
6: **if** $\mathsf{P} = Q$ **then**
7: $\quad \mathsf{cnt}_Q^{\mathsf{send}} \leftarrow \mathsf{cnt}_Q^{\mathsf{send}} + 1$
8: $\quad$ **if** $\mathsf{cnt}_Q^{\mathsf{send}} \leqslant m - 1$ **then**
9: $\quad\quad \mathsf{cnt} \leftarrow \mathsf{cnt}_Q^{\mathsf{send}} + 1$
10: $\quad\quad \mathsf{sk}' \leftarrow \mathsf{k}_{\mathsf{cnt}}$
11: $\mathsf{st}_\mathsf{P} \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{sk}', \mathsf{rk})$
12: append $(\mathsf{ad}_\mathsf{P}, \mathsf{pt}_\mathsf{P})$ to $\mathsf{sent}_{\mathsf{pt}}^{\mathsf{P}}$
13: append $(\mathsf{ad}_\mathsf{P}, \mathsf{ct}_\mathsf{P})$ to $\mathsf{sent}_{\mathsf{ct}}^{\mathsf{P}}$
14: **return** ct_P

Oracle RATCH (P, “rec”, ad, ct)

1: $\mathsf{ct}_\mathsf{P} \leftarrow \mathsf{ct}$
2: $\mathsf{ad}_\mathsf{P} \leftarrow \mathsf{ad}$
3: parse $\mathsf{st}_\mathsf{P} = (1^\lambda, \mathsf{hk}, \mathsf{sk}, \mathsf{rk})$
4: $\mathsf{pt}_\mathsf{P} \leftarrow \mathrm{OTAE.Dec}(\mathsf{rk}, \mathsf{ad}, \mathsf{ct})$
5: **if** $\mathsf{pt}_\mathsf{P} = \perp$ **then**
6: $\quad$ **return** false
7: $\mathsf{rk}' \leftarrow \mathrm{H.Eval}(\mathsf{hk}, \mathsf{rk})$
8: **if** $\mathsf{P} = \overline{Q}$ **then**
9: $\quad \mathsf{cnt}_{\overline{Q}}^{\mathsf{rec}} \leftarrow \mathsf{cnt}_{\overline{Q}}^{\mathsf{rec}} + 1$
10: $\quad$ **if** $\mathsf{cnt}_{\overline{Q}}^{\mathsf{rec}} \leqslant m - 1$ **then**
11: $\quad\quad \mathsf{cnt} \leftarrow \mathsf{cnt}_{\overline{Q}}^{\mathsf{rec}} + 1$
12: $\quad\quad \mathsf{rk}' \leftarrow \mathsf{k}_{\mathsf{cnt}}$
13: $\mathsf{st}_{\overline{\mathsf{P}}} \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{sk}, \mathsf{rk}')$
14: append $(\mathsf{ad}_\mathsf{P}, \mathsf{pt}_\mathsf{P})$ to $\mathsf{received}_{\mathsf{pt}}^{\mathsf{P}}$
15: append $(\mathsf{ad}_\mathsf{P}, \mathsf{ct}_\mathsf{P})$ to $\mathsf{received}_{\mathsf{ct}}^{\mathsf{P}}$
16: **return** true

Oracle $\mathsf{EXP}_{\mathsf{st}}(\mathsf{P})$

1: **if** $(\mathsf{P} = Q$ and $\mathsf{cnt}_Q^{\mathsf{send}} \leqslant n)$ or $(\mathsf{P} = \overline{Q}$ and $\mathsf{cnt}_{\overline{Q}}^{\mathsf{rec}} \leqslant n)$ **then** abort
2: **return** st_P

Oracle $\mathsf{EXP}_{\mathsf{pt}}(\mathsf{P})$

1: **return** pt_P

Fig. 9. FORGE-Security: Hybrids $\Gamma_{Q,m,n}$

$\mathcal{E}^{\mathsf{ENC},\mathsf{DEC}}()$:

1: $\mathsf{hk} \xleftarrow{\$} \mathrm{H}.\mathcal{K}(\lambda)$
2: $\mathsf{cnt}^{\mathsf{send}}_{\mathrm{Q}}, \mathsf{cnt}^{\mathsf{rec}}_{\overline{\mathrm{Q}}} \leftarrow 0$
3: pick $\mathsf{k}_1, \ldots, \mathsf{k}_{n-1}, \mathsf{k}_{n+1}, \mathsf{k} \xleftarrow{\$} \mathcal{K}_\lambda$, $\mathsf{k}_n \leftarrow \perp$
4: $\mathsf{st}_{\mathrm{Q}} \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{k}_1, \mathsf{k})$
5: $\mathsf{st}_{\overline{\mathrm{Q}}} \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{k}, \mathsf{k}_1)$
6: set all sent^*_* and $\mathsf{received}^*_*$ variables to $\emptyset$
7: $(\mathrm{P}, \mathsf{ad}^*, \mathsf{ct}^*) \leftarrow \mathcal{A}^{\mathsf{RATCH},\mathsf{EXP}_{\mathsf{st}},\mathsf{EXP}_{\mathsf{pt}}}(\mathsf{hk})$
8: **if** $\mathrm{P} \neq \overline{\mathrm{Q}}$ or $\mathsf{cnt}^{\mathsf{send}}_{\mathrm{Q}} \neq n$ **then** abort
9: **return** $(\mathsf{ad}^*, \mathsf{ct}^*)$

Subroutine RATCH (P, "send", ad, pt)

1: $\mathsf{pt}_{\mathrm{P}} \leftarrow \mathsf{pt}$
2: $\mathsf{ad}_{\mathrm{P}} \leftarrow \mathsf{ad}$
3: parse $\mathsf{st}_{\mathrm{P}} = (1^\lambda, \mathsf{hk}, \mathsf{sk}, \mathsf{rk})$
4: **if** $\mathrm{P} = \mathrm{Q}$ and $\mathsf{cnt}^{\mathsf{send}}_{\mathrm{Q}} = n - 1$ **then**
5: $\quad \mathsf{ct}_{\mathrm{P}} \leftarrow \mathsf{ENC}(\mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$
6: $\quad \mathsf{cnt}^{\mathsf{send}}_{\mathrm{Q}} = \mathsf{cnt}^{\mathsf{send}}_{\mathrm{Q}} + 1$
7: $\quad \mathsf{sk}' \leftarrow \mathsf{k}_{n+1}$
8: **else**
9: $\quad \mathsf{ct}_{\mathrm{P}} \leftarrow \mathrm{OTAE.Enc}(\mathsf{sk}, \mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$
10: $\quad \mathsf{sk}' \leftarrow \mathrm{H.Eval}(\mathsf{hk}, \mathsf{sk})$
11: $\quad$ **if** $\mathrm{P} = \mathrm{Q}$ **then**
12: $\quad\quad \mathsf{cnt}^{\mathsf{send}}_{\mathrm{Q}} = \mathsf{cnt}^{\mathsf{send}}_{\mathrm{Q}} + 1$
13: $\quad\quad$ **if** $\mathsf{cnt}^{\mathsf{send}}_{\mathrm{Q}} \leqslant n - 1$ **then**
14: $\quad\quad\quad \mathsf{cnt} \leftarrow \mathsf{cnt}^{\mathsf{send}}_{1} + 1$
15: $\quad\quad\quad \mathsf{sk}' \leftarrow \mathsf{k}_{\mathsf{cnt}}$
16: $\mathsf{st}_{\mathrm{P}} \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{sk}', \mathsf{rk})$
17: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$ to $\mathsf{sent}^{\mathrm{P}}_{\mathsf{pt}}$
18: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{ct}_{\mathrm{P}})$ to $\mathsf{sent}^{\mathrm{P}}_{\mathsf{ct}}$
19: **return** ct_{P}

Subroutine RATCH (P, "rec", ad, ct)

1: $\mathsf{ct}_{\mathrm{P}} \leftarrow \mathsf{ct}$
2: $\mathsf{ad}_{\mathrm{P}} \leftarrow \mathsf{ad}$
3: parse $\mathsf{st}_{\mathrm{P}} = (1^\lambda, \mathsf{hk}, \mathsf{sk}, \mathsf{rk})$
4: **if** $\mathrm{P} = \overline{\mathrm{Q}}$ and $\mathsf{cnt}^{\mathsf{rec}}_{\overline{\mathrm{Q}}} = n - 1$ **then**
5: $\quad$ abort
6: **else**
7: $\quad \mathsf{pt}_{\mathrm{P}} \leftarrow \mathrm{OTAE.Dec}(\mathsf{rk}, \mathsf{ad}, \mathsf{ct})$
8: $\quad$ **if** $\mathsf{pt}_{\mathrm{P}} = \perp$ **then**
9: $\quad\quad$ **return** false
10: $\quad \mathsf{rk}' \leftarrow \mathrm{H.Eval}(\mathsf{hk}, \mathsf{rk})$
11: $\quad$ **if** $\mathrm{P} = \overline{\mathrm{Q}}$ **then**
12: $\quad\quad \mathsf{cnt}^{\mathsf{rec}}_{\overline{\mathrm{Q}}} \leftarrow \mathsf{cnt}^{\mathsf{rec}}_{\overline{\mathrm{Q}}} + 1$
13: $\quad\quad$ **if** $\mathsf{cnt}^{\mathsf{rec}}_{\overline{\mathrm{Q}}} \leqslant n - 1$ **then**
14: $\quad\quad\quad \mathsf{cnt} \leftarrow \mathsf{cnt}^{\mathsf{rec}}_{\overline{\mathrm{Q}}} + 1$
15: $\quad\quad\quad \mathsf{rk}' \leftarrow \mathsf{k}_{\mathsf{cnt}}$
16: $\mathsf{st}_{\overline{\mathrm{P}}} \leftarrow (1^\lambda, \mathsf{hk}, \mathsf{sk}, \mathsf{rk}')$
17: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$ to $\mathsf{received}^{\mathrm{P}}_{\mathsf{pt}}$
18: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{ct}_{\mathrm{P}})$ to $\mathsf{received}^{\mathrm{P}}_{\mathsf{ct}}$
19: **return** true

Subroutine $\mathsf{EXP}_{\mathsf{st}}(\mathrm{P})$

1: **if** $(\mathrm{P} = \mathrm{Q}$ and $\mathsf{cnt}^{\mathsf{send}}_{\mathrm{Q}} \leqslant n)$ or $(\mathrm{P} = \overline{\mathrm{Q}}$ and $\mathsf{cnt}^{\mathsf{rec}}_{\overline{\mathrm{Q}}} \leqslant n)$ **then** abort
2: **return** st_{P}

Subroutine $\mathsf{EXP}_{\mathsf{pt}}(\mathrm{P})$

1: **return** pt_{P}

Fig. 10. Adversary $\mathcal{E}$: Simulating $\Gamma_{\mathrm{Q},n+1,n}$

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.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 129–158. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_5
2. Balli, F., Rösler, P., Vaudenay, S.: Determining the core primitive for optimally secure ratcheting. IACR Cryptology ePrint Archive 2020/148 (2020)
3. Bellare, M., Singh, A.C., Jaeger, J., Nyayapati, M., Stepanovs, I.: Ratcheted encryption and key exchange: the security of messaging. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 619–650. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_21
4. Blazy, O., Bossuat, A., Bultel, X., Fouque, P., Onete, C., Pagnin, E.: SAID: reshaping signal into an identity-based asynchronous messaging protocol with authenticated ratcheting. In: EuroS&P, pp. 294–309. IEEE (2019)
5. Borisov, N., Goldberg, I., Brewer, E.A.: Off-the-record communication, or, why not to use PGP. In: WPES, pp. 77–84. ACM (2004)
6. Caforio, A., Durak, F.B., Vaudenay, S.: On-demand ratcheting with security awareness. IACR Cryptology ePrint Archive 2019/965 (2019). https://eprint.iacr.org/2019/965
7. Cohn-Gordon, K., Cremers, C.J.F., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the signal messaging protocol. In: EuroS&P, pp. 451–466. IEEE (2017)
8. Durak, F.B., Vaudenay, S.: Bidirectional asynchronous ratcheted key agreement with linear complexity. In: Attrapadung, N., Yagi, T. (eds.) IWSEC 2019. LNCS, vol. 11689, pp. 343–362. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26834-3_20
9. Jaeger, J., Stepanovs, I.: Optimal channel security against fine-grained state compromise: the safety of messaging. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 33–62. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_2
10. Jost, D., Maurer, U., Mularczyk, M.: Efficient ratcheting: almost-optimal guarantees for secure messaging. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 159–188. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_6
11. Jost, D., Maurer, U., Mularczyk, M.: A unified and composable take on ratcheting. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11892, pp. 180–210. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36033-7_7
12. Perrin, T., Marlinspike, M.: The double ratchet algorithm. GitHub wiki (2016)
13. Poettering, B., Rösler, P.: Towards bidirectional ratcheted key exchange. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 3–32. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_1
14. Systems, O.W.: Signal protocol library for Java/Android. GitHub repository (2017). https://github.com/WhisperSystems/libsignal-protocol-java