https://doi.org/10.1093/comjnl/bxab209

Optimal Symmetric Ratcheting for Secure Communication

Hailun Yan[1,2,*], Serge Vaudenay[2], Daniel Collins[2] and Andrea Caforio[2]

[1]*University of Chinese Academy of Sciences, Beijing 100049, China*
[2]*École Polytechnique Fédérale de Lausanne (EPFL), Lausanne 1015, Switzerland*
[*]*Corresponding author: yanhl.crypt@gmail.com*

To mitigate state exposure threats to long-lived instant messaging sessions, ratcheting was introduced, which is used in practice in protocols like Signal. However, existing ratcheting protocols generally come 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 so-called ratcheting on-demand. The light protocol they proposed has still a high complexity.
In this paper, we propose the lightest possible protocol we could imagine, which essentially encrypts and then hashes the secret key. We prove it secure in the standard model by introducing a new security notion, which relates symmetric encryption with key updates by hashing. Our protocol composes well with the generic transformation techniques by Caforio et al. to offer high security and performance at the same time. In a second step, we propose another protocol based on a newly defined integrated primitive, extending standard one-time authenticated encryption with an additional output block used as a secret key for the next message. We instantiate this primitive firstly from any authenticated encryption with associated data, and then we propose an efficient instantiation using advanced encryption standard (AES) encryption to update the key and AES-Galois/Counter mode of operation to encrypt and decrypt messages.

Keywords: secure communication; ratcheting; forward secrecy; standard model

Received 5 May 2021; Revised 14 September 2021; Editorial Decision 17 November 2021
Handling editor: Xinyi Huang

1. INTRODUCTION

Classic communication models usually assume that endpoints are secure while the adversary is on the communication channel. However, protocols in recent messaging applications regularly update (or ratchet) the keying material due to practical attack vectors like malware and system vulnerabilities, which can lead to state exposures. One notable example of ratcheting is in the Signal protocol [1] by Open Whisper Systems, which uses the Double Ratchet algorithm [2] once initial keys are established. It is also widely used in WhatsApp [3], Skype [4] and other secure messaging systems [5].

Ratcheting is an umbrella term for a number of different security guarantees. The 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 iterating a one-way function and deleting previous states. Moreover, to prevent the adversary from decrypting messages or injecting ciphertexts in the future, randomness is used when updating every state to obtain so-called *post-compromise security* [6]. Ratcheting is mainly related to how keys are used and updated, rather than how they are obtained [7]. We thereby will 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 roles as senders and receivers. 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 not known in advance. One interesting notion introduced by Alwen *et al.* [8] is that of *immediate decryption*. Here, users can decrypt messages out-of-order and continue protocol execution when messages are lost without disruption. This property is beneficial when the communication

network is unreliable. Adding reliability to the communication channel can indeed be solved by a lower-level protocol. Hence, we *do not* consider immediate decryption in our constructions.

Previous work. Forward secrecy was introduced as design goal for key exchange in the 1980s [9] and has since been considered both in protocols and standalone primitives like public-key encryption and signatures. Notably, Bellare and Yee [10] examine different constructions of forward-secret symmetric primitives, but do not consider message exchange. For post-compromise security, the ratcheting technique using asymmetric cryptography was first deployed in the off-the-record [11] messaging protocol and the Signal protocol [1]. A clean description of Signal was published by designers Marlinspike and Perrin [2]. Cohn-Gordon *et al.* [12] later gave the first academic analysis of Signal's security, which, in particular, considered Signal's key exchange mechanism which makes use of public-key infrastructure (PKI). In [13], Blazy et al. note attacks (outside of Cohn-Gordon *et al.*'s model) on Signal and propose an identity-based asynchronous messaging protocol with explicit authentication albeit which relies on a trusted identity manager (rather than PKI). At EUROCRYPT 2019, Alwen *et al.* [8] modularized Signal and proposed two ratcheting protocols (denoted as ACD and ACD-PK) with security against adversarially chosen random coins, both of which support immediate decryption. As in our work, their protocols assume that an initial key exchange between given parties was successful. Following Alwen et al., we consider Signal and ACD as equivalent.

The first formal definitions concerning ratcheting were given by Bellare *et al.* [7] at CRYPTO 2017. Following their work, papers considering different security levels, properties and underlying primitives have been written [8, 14–18]. Some of these results consider secure messaging while others are about key exchange. These works can be considered as studying equivalent notions since secure ratcheted communication can be reduced to secure ratcheted key exchange (RKE). At CRYPTO 2018, Poettering and Rösler [14] designed a protocol with 'optimal' security, in that security is only broken by attacks that cannot be prevented by any protocol satisfying their primitive definition. However, they require random oracles and heavy primitives like hierarchical identity-based encryption (HIBE). At the same conference, Jaeger and Stepanovs [15] proposed a similar protocol with security against compromised random coins—namely, with random coin leakage before usage—which also requires HIBE. Durak and Vaudenay (DV) [18] proposed a protocol with slightly lower security without using HIBE or random oracles. They also prove that public-key encryption is necessary to achieve meaningful post-compromise security. At EUROCRYPT 2019, Jost *et al.* [17] proposed a protocol with security between optimal security and the security of the DV protocol. Jost *et al.* [19] modelled existing ratcheting protocols in the so-called Constructive Cryptography framework with the aim of modularizing and unifying existing definitions. Balli *et al.* [20] modelled optimally secure RKE under randomness manipulation and showed that a so-called key-updatable key encapsulation mechanism (which has only been constructed from HIBE so far) is a necessary and sufficient building block when only one party sends and the other receives.

Caforio *et al.* [16] adapted the work of DV [18] to define asynchronous ratcheted communication with associated data (ARCAD). Moreover, they defined a *hybrid* bidirectional secure communication protocol to enable *on-demand ratcheting*: By integrating two ratcheting protocols with different security levels—a strongly secure protocol (such as the DV protocol) and a weaker but lighter protocol—the hybrid system allows the sender to select which security level he wants to use. When ratcheting becomes infrequent, the communication system enjoys satisfactory implementation performance thanks to the underlying light protocol. For this, they designed a weakly secure protocol called liteARCAD, which is based solely on symmetric cryptography. It achieves provable forward secrecy and strong software performance, albeit it does not provide post-compromise security. They further generically strengthen protocols by introducing the notion of *security awareness*.

Although already quite efficient, liteARCAD still has high complexities for sending and receiving 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 operations per message is still high: sending a message requires one hash and $n+1$ symmetric encryptions. Receiving a message is similar.

Note that Signal/ACD [1, 8] combine asymmetric and symmetric cryptography in a hybrid-like construction, but do not allow each component to be used on-demand. Instead, an asymmetric ratchet is iterated when the direction of communication changes. When the direction of communication does not change, the ACD protocol is fully based on symmetric cryptography (a so-called FS-authenticated encryption with associated data (AEAD), which is instantiated from an AEAD and a pseudorandom generator (PRG)).

Contribution. In this work, 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 that relates symmetric encryption with key updates by hashing. Essentially, we require 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 satisfies the same security as liteARCAD: informally, EtH satisfies forward security, confidentiality and authentication. We deduce that we can use EtH in the generic constructions of Caforio *et al.* [16] and achieve the same security and superior performance compared with liteARCAD.

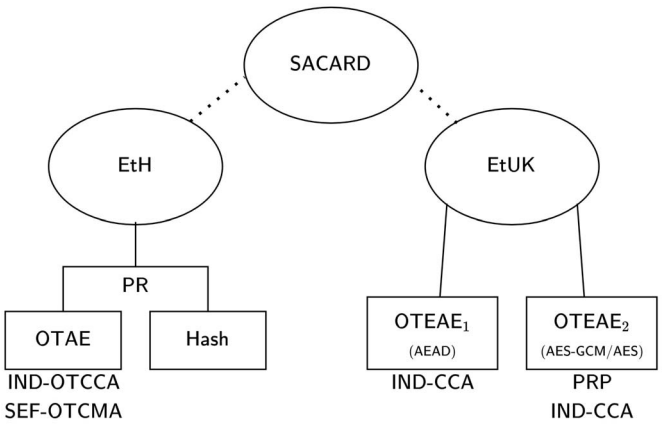


FIGURE 1. A summary of our contributions: our primitive SARCAD and two instantiations EtH and EtUK (circled). Boxed symbols refer to the underlying primitives used to build the respective SARCAD. Unboxed symbols refer to assumptions used to prove the security of the corresponding SARCAD.

We also give a proposal called *Encrypt-then-Update-Key* (EtUK), which is comparable to EtH. It is based on a newly introduced integrated primitive, extending standard one-time authenticated-encryption (OTAE) with an additional output block used as a secret key for the next session. We formally define this primitive, named one-time extended authenticated-encryption (OTEAE), which may be of independent interest. We give a generic construction of OTEAE from standard AEAD. We also propose an optimized OTEAE construction based on advanced encryption standard (AES)-Galois/Counter mode of operation (GCM) under the assumption that AES is a pseudo-random (PR) permutation. Owing to the popularity of AES, we can thus take advantage of efficient hardware and software implementations in practice. More generally, the two constructions EtH and EtUK admit different performance characteristics and allow for different instantiations as required by context.

Differences between proceedings version and this paper. We note that a shorter conference version of this paper was presented at the 15th International Workshop on Security in Yan and Vaudenay (2020) [21]. In the conference paper, they introduced the symmetric-cryptography-based asynchronous ratcheted communication with associated data (SARCAD) construction EtH and proved it secure. However, they did not explore different instantiations of SARCAD based on different hardness assumptions and primitives. To this end, our main additional contributions in this work are as follows:

- We define another SARCAD construction, namely EtUK, based on the new primitive OTEAE. We prove that EtUK is secure.
- We instantiate the OTEAE primitive in two ways and prove that both constructions are secure.
- We provide preliminary benchmarks for our constructions and previous ratcheting protocols and demonstrate the efficiency of our protocols.

Organization. In Section 2, we revisit the preliminary notions from [16], which are relevant to our work. In Section 3, we construct a correct and secure SARCAD protocol. Meanwhile, we define a new security notion with respect to OTAE and a hash function family. We formally prove that our scheme is secure. In Section 4, we propose another SARCAD protocol based on OTEAE, which we formalize. In Section 5, we give two constructions of OTEAE and prove their security. In Section 6, we implement our protocols and evaluate their performance. Finally, we conclude in Section 7.

2. PRIMITIVES

2.1. Notations and general definitions

We consider two participants Alice (A) and Bob (B). Whenever we talk about either participant, we represent it as P; $\overline{P}$ then refers to P's counterpart. Parties 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 messaging), we call them *sender S* and *receiver R*.

Let λ, a positive integer, be a security parameter hereafter. An algorithm is either deterministic (of the form $F(x) \to y$) or probabilistic (of the form $F(x) \xrightarrow{\$} y$), where in the latter case, we assume the use of uniformly sampled randomness. We use Pascal case when referring to algorithms from a given primitive (e.g. AEAD.Enc), and UPPER case to refer to oracles in security games/experiments (e.g. ENC).

Hash function family. A hash function family H is defined by a key space $H.\mathcal{K}(\lambda)$, a domain $H.\mathcal{D}(\lambda)$, and a polynomially bounded deterministic algorithm $H.\mathsf{Eval}(\mathsf{hk}, \mathsf{x})$, which takes a key hk in $H.\mathcal{K}(\lambda)$ and a bitstring x in $H.\mathcal{D}(\lambda)$ to produce a digest in $H.\mathcal{D}(\lambda)$.

OTAE. In practice, one often uses a symmetric key to encrypt a single message. The key is never used again. In these settings, one can use a OTAE with associated data scheme.

An OTAE is defined by 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 $\mathcal{K}(\lambda)$, 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 $\perp$. For correctness, OTAE satisfies

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

for all $\mathsf{k} \in \mathsf{OTAE}.\mathcal{K}(\lambda)$, and $\mathsf{ad}, \mathsf{pt} \in \mathsf{OTAE}.\mathcal{D}(\lambda)$. OTAE satisfies one-time indistinguishable under chosen ciphertext attack (IND-CCA) security and one-time srongly existentially unforgeable under chosen message attack (SEF-CMA) security, defined as follows.

DEFINITION 2.1. *(IND-OTCCA Security for* OTAE [16]*). An* OTAE *scheme is* (q, ϵ)-IND-OTCCA*-secure, if for any PPT adversary* $\mathcal{A}$ *bounded by* $q(\lambda)$ *queries to oracles, the advantage*

$$Adv(\mathcal{A}) = Pr\left[\mathsf{IND} - \mathsf{OTCCA}_0^{\mathcal{A}}(1^\lambda) \to 1\right] - Pr\left[\mathsf{IND} - \mathsf{OTCCA}_1^{\mathcal{A}}(1^\lambda) \to 1\right]$$

of $\mathcal{A}$ *playing the following game is bounded by* $\epsilon(\lambda)$.

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

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

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

DEFINITION 2.2. *(SEF-OTCMA Security for* OTAE [16]*). An* OTAE *scheme is* (q, ϵ)-SEF-OTCMA*-secure if for any PPT adversary* $\mathcal{A}$ *bounded by* $q(\lambda)$ *queries to oracles, the advantage* $Pr\left[\mathsf{SEF} - \mathsf{OTCMA}^{\mathcal{A}}(1^\lambda) \to 1\right]$ *of* $\mathcal{A}$ *playing the following game is bounded by* ϵ.

Game $\mathsf{SEF\text{-}OTCMA}^{\mathcal{A}}(1^\lambda)$:
1: $\mathsf{k} \leftarrow \mathsf{OTAE}.\mathcal{K}(\lambda)$
2: $\mathsf{query}_{\mathsf{enc}} \leftarrow \perp$
3: $\mathcal{A}^{\mathsf{ENC}}(1^\lambda) \to (\mathsf{ad}', \mathsf{ct}')$
4: **if** $(\mathsf{ad}', \mathsf{ct}') = \mathsf{query}_{\mathsf{enc}}$ **then** abort
5: **end if**
6: **if** $\mathsf{OTAE.Dec}(\mathsf{k}, \mathsf{ad}', \mathsf{ct}') = \perp$ **then** abort
7: **end if**
8: **return** 1

Oracle $\mathsf{ENC}(\mathsf{ad}, \mathsf{pt})$
1: **if** $\mathsf{query}_{\mathsf{enc}} \neq \perp$ **then** abort
2: **end if**
3: $\mathsf{OTAE.Enc}(\mathsf{k}, \mathsf{ad}, \mathsf{pt}) \to \mathsf{ct}$
4: $\mathsf{query}_{\mathsf{enc}} \leftarrow (\mathsf{ad}, \mathsf{ct})$
5: **return** ct

AEAD. AEAD [22] follows the same syntax and correctness definition as OTAE. We require the following IND-CCA security notion, which allows the adversary to call the encryption oracle many times (in contrast to OTAE).

DEFINITION 2.3. *(*IND-CCA *Security for* AEAD*). An* AEAD *scheme is* (q, ϵ)-IND-CCA*-secure, if for any PPT adversary* $\mathcal{A}$ *bounded by* $q(\lambda)$ *queries to oracles, the advantage*

$$Adv(\mathcal{A}) = Pr\left[\mathsf{IND} - \mathsf{CCA}_0^{\mathcal{A}}(1^\lambda) \to 1\right] - Pr\left[\mathsf{IND} - \mathsf{CCA}_1^{\mathcal{A}}(1^\lambda) \to 1\right]$$

of $\mathcal{A}$ *playing the following game is bounded by* $\epsilon(\lambda)$.

Game $\mathsf{IND\text{-}CCA}_b^{\mathcal{A}}(1^\lambda)$:
1: challenge $\leftarrow \perp$
2: $\mathsf{k} \xleftarrow{\$} \mathsf{AEAD}.\mathcal{K}(\lambda)$
3: $\mathcal{A}^{\mathsf{ENC},\mathsf{DEC}}(1^\lambda) \to b'$
4: **return** b'

Oracle $\mathsf{ENC}(\mathsf{ad}, \mathsf{pt})$
1: **if** $\mathsf{b} = 0$ **then**
2: replace pt by a random message of the same length
3: **end if**
4: $\mathsf{AEAD.Enc}(\mathsf{k}, \mathsf{ad}, \mathsf{pt}) \to \mathsf{ct}$
5: challenge $=$ challenge $\cup \{(\mathsf{ad}, \mathsf{ct})\}$
6: **return** ct

Oracle $\mathsf{DEC}(\mathsf{ad}, \mathsf{ct})$
1: **if** $(\mathsf{ad}, \mathsf{ct}) \in$ challenge **then return** $\perp$
2: **end if**
3: **if** $b = 0$ **then return** $\perp$
4: **end if**
5: **return** $\mathsf{AEAD.Dec}(\mathsf{k}, \mathsf{ad}, \mathsf{ct})$

2.2. Definition of SARCAD

We slightly adapt the definition of ARCAD from Caforio *et al.* [16] for SARCAD.

DEFINITION 2.4. *(*SARCAD*). A symmetric-cryptography-based ARCAD (*SARCAD*) consists of the following polynomial-time algorithms:*

- $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$*: This probabilistic algorithm defines the common public parameters* pp.
- $\mathsf{Initall}(\mathsf{pp}) \xrightarrow{\$} (\mathsf{st}_A, \mathsf{st}_B)$*: This probabilistic algorithm returns the initial state of A and B.*

- $\mathsf{Send}(\mathsf{st}_P, \mathsf{ad}, \mathsf{pt}) \rightarrow (\mathsf{st}'_P, \mathsf{ct})$*: This deterministic algorithm takes as input a plaintext* pt *and associated data* ad *and produces a ciphertext* ct *together with an updated state* st'_P.
- $\mathsf{Receive}(\mathsf{st}_P, \mathsf{ad}, \mathsf{ct}) \rightarrow (\mathsf{acc}, \mathsf{st}'_P, \mathsf{pt})$*: This deterministic algorithm takes as input a ciphertext* ct *and associated data* ad *and produces a plaintext* pt *together with an updated state* st'_P *and a flag* acc.

In our work, we assume that $\mathsf{acc} = \mathsf{false}$ in $\mathsf{Receive}$ implies that $\mathsf{st}'_P = \mathsf{st}_P$ and $\mathsf{pt} = \perp$. We do not update the state when reception fails to mitigate Denial-of-Service attacks. If message reception erased the state upon failure, the adversary would not have the opportunity to make several decryption attempts, and so the model would be weaker.

Informally, a SARCAD protocol functions correctly if the receiver performs a decryption of an honestly sent ciphertext and outputs the same plaintext as sent by its counterpart. Correctness is only required when participant P receives messages in the same order as those sent by participant $\overline{P}$.

We formally define the $\mathsf{CORRECTNESS}$ game in Fig. 2. In this game, we initialize two participants. 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 $\overline{P}$. The scheduling[1] is defined by a sequence sched of tuples of the form either $(P, \text{`}\mathsf{send}', \cdot, \cdot)$ or $(P, \text{`}\mathsf{rec}')$. We model sending/receiving by calls of the form $\mathsf{RATCH}(P, \text{`}\mathsf{role}', \cdot, \cdot)$ as depicted in Fig. 2. Meanwhile, we define variables $\mathsf{sent}^P_{\mathsf{pt}}$ (resp. $\mathsf{received}^P_{\mathsf{pt}}$) which keep a list of messages sent (resp. received) by participant P when running Send (resp. $\mathsf{Receive}$).

For each variable v such as $\mathsf{sent}^P_{\mathsf{pt}}$ or st_P relative to participant P, we denote by $v(t)$ the value of v at time t. Note that the notion of *time* is participant-specific, which refers to the number of elementary operations the participant has performed. We assume that security games implicitly track time for participants. We emphasize that we assume neither synchronization nor a central clock: we use this notion of time in order to define security.

DEFINITION 2.5. *(Correctness* [16]*). We say that a* SARCAD *protocol is correct if for all sequences* sched *of tuples of the form* $(P, \text{`}\mathsf{send}', \mathsf{ad}, \mathsf{pt})$ *and* $(P, \text{`}\mathsf{rec}')$*, the game in Fig. 2 never returns 1. Namely,*

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

Note that this definition has been simplified from the conference paper [21], which considered sched_i values of the form $(P, \text{`}\mathsf{rec}', \mathsf{ad}, \mathsf{ct})$ which led to ambiguity.

To formally capture the status of parties during communication, Caforio *et al.* [16] gave a set of intuitive definitions (originally defined in [18] and adapted to ARCAD). For completeness, we define notions necessary for our analysis below. For more details, please refer to [16] and [18].

DEFINITION 2.6. *(Matching status* [16]*). We say that* P *is in a matching status in a given game at time* t *if*

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

DEFINITION 2.7. *(Forgery* [16]*). Given a participant* P *in a game, we say that* $(\mathsf{ad}, \mathsf{ct}) \in \mathsf{received}^P_{\mathsf{ct}}$ *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 is no longer after receiving* $(\mathsf{ad}, \mathsf{ct})$.

2.3. Security of SARCAD

We define the security of SARCAD with an $\mathsf{IND\text{-}CCA}$ notion resp. FORGE security, which is captured by the advantage of an adversary playing the $\mathsf{IND\text{-}CCA}$ resp. FORGE game.

In both games, the adversary can call RATCH oracles (Fig. 2) in addition to several others called $\mathsf{EXP}_{\mathsf{st}}$, $\mathsf{EXP}_{\mathsf{pt}}$ and $\mathsf{CHALLENGE}$ (Fig. 3). All those oracles are used without change in all relevant security notions in this paper.

- RATCH. This oracle is used to either to send or to receive, which captures message exchange.
- $\mathsf{EXP}_{\mathsf{st}}$. This oracle is used to obtain the state of a given participant.
- $\mathsf{EXP}_{\mathsf{pt}}$. This oracle is used to obtain the last received message of a participant.
- $\mathsf{CHALLENGE}$. This oracle is used (only in the $\mathsf{IND\text{-}CCA}$ game) to send either an input plaintext or a random string.

Following previous work [16, 18], we introduce the notion of a *cleanness* predicate when defining the security of a SARCAD scheme. A cleanness predicate C_{clean} identifies and captures all trivial ways of attacking a scheme. The cleanness predicate prevents the adversary from making such trivial attacks without losing the corresponding game and is checked at Line 6 of the $\mathsf{IND\text{-}CCA}$ and FORGE games in Fig. 3.

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

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

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

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

Game CORRECTNESS(1^λ, sched):
1: set all sent^*_* and $\mathsf{received}^*_*$ to $\emptyset$
2: $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$
3: $\mathsf{Initall}(\mathsf{pp}) \xrightarrow{\$} (\mathsf{st_A}, \mathsf{st_B})$
4: initialize FIFO lists $\mathsf{incoming_A}, \mathsf{incoming_B} \leftarrow \emptyset$
5: $\mathsf{i} \leftarrow 0$
6: **loop**
7: $\quad \mathsf{i} \leftarrow \mathsf{i} + 1$
8: $\quad$ **if** $\mathsf{sched_i}$ of form (P, "rec") **then**
9: $\quad\quad$ **if** $\mathsf{incoming_P}$ is empty **then**
10: $\quad\quad\quad$ **return** 0
11: $\quad\quad$ pull (ad, ct) from $\mathsf{incoming_P}$
12: $\quad\quad$ acc $\leftarrow \mathsf{RATCH}(\mathsf{P}, \text{"rec"}, \mathsf{ad}, \mathsf{ct})$
13: $\quad\quad$ **if** acc = false **then return** 1
14: $\quad$ **else**
15: $\quad\quad$ parse $\mathsf{sched_i} = (\mathsf{P}, \text{"send"}, \mathsf{ad}, \mathsf{pt})$
16: $\quad\quad$ ct $\leftarrow \mathsf{RATCH}(\mathsf{P}, \text{"send"}, \mathsf{ad}, \mathsf{pt})$
17: $\quad\quad$ push (ad, ct) to $\mathsf{incoming_{\bar{P}}}$
18: $\quad$ **if** $\mathsf{received}^{\mathsf{A}}_{\mathsf{pt}}$ not prefix of $\mathsf{sent}^{\mathsf{B}}_{\mathsf{pt}}$ **then**
19: $\quad\quad$ **return** 1
20: $\quad$ **if** $\mathsf{received}^{\mathsf{B}}_{\mathsf{pt}}$ not prefix of $\mathsf{sent}^{\mathsf{A}}_{\mathsf{pt}}$ **then**
21: $\quad\quad$ **return** 1

FIGURE 2. Definition of RATCH oracles and the CORRECTNESS game for a SARCAD protocol.

Oracle $\mathsf{EXP_{st}}(\mathsf{P})$
1: **return** $\mathsf{st_P}$

Oracle $\mathsf{EXP_{pt}}(\mathsf{P})$
1: **return** $\mathsf{pt_P}$

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

Game $\mathsf{IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{b},\mathsf{C_{clean}}}(1^\lambda)$:
1: $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$
2: $\mathsf{Initall}(\mathsf{pp}) \xrightarrow{\$} (\mathsf{st_A}, \mathsf{st_B})$
3: set all sent^*_* and $\mathsf{received}^*_*$ variables to $\emptyset$
4: set $\mathsf{t_{test}}$ to $\perp$
5: $\mathsf{b'} \leftarrow \mathcal{A}^{\mathsf{RATCH},\mathsf{EXP_{st}},\mathsf{EXP_{pt}},\mathsf{CHALLENGE}}(\mathsf{pp})$
6: **if** $\neg \mathsf{C_{clean}}$ **then return** $\perp$
7: **return** b′

Game $\mathsf{FORGE}^{\mathcal{A}}_{\mathsf{C_{clean}}}(1^\lambda)$:
1: $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$
2: $\mathsf{Initall}(\mathsf{pp}) \xrightarrow{\$} (\mathsf{st_A}, \mathsf{st_B})$
3: set all sent^*_* and $\mathsf{received}^*_*$ variables to $\emptyset$
4: $(\mathsf{P}, \mathsf{ad}^*, \mathsf{ct}^*) \leftarrow \mathcal{A}^{\mathsf{RATCH},\mathsf{EXP_{st}},\mathsf{EXP_{pt}}}(\mathsf{pp})$
5: $\mathsf{RATCH}(\mathsf{P}, \text{"rec"}, \mathsf{ad}^*, \mathsf{ct}^*) \rightarrow \mathsf{acc}$
6: **if** $\neg \mathsf{C_{clean}}$ **then return** 0
7: **if** acc = false **then return** 0
8: **if** $(\mathsf{ad}^*, \mathsf{ct}^*)$ is not a forgery (Def. 2.7) for P **then**
9: $\quad$ **return** 0
10: **return** 1

FIGURE 3. IND-CCA, FORGE games.

DEFINITION 2.8. *(C_{clean}-IND-CCA Security [16]). Given a cleanness predicate C_{clean}, consider the $\mathit{IND\text{-}CCA}^{\mathcal{A}}_{b,C_{clean}}$ game in Fig. 3. We say that* SARCAD *is (q, ϵ)-C_{clean}-IND-CCA-secure if for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ bounded by $q(\lambda)$ queries to oracles, the advantage*

$$\mathit{Adv}(\mathcal{A}) = \left| Pr\left[\mathit{IND-CCA}^{\mathcal{A}}_{0,C_{clean}}(1^\lambda) \rightarrow 1\right] - Pr\left[\mathit{IND-CCA}^{\mathcal{A}}_{1,C_{clean}}(1^\lambda) \rightarrow 1\right]\right|$$

of $\mathcal{A}$ in $\mathit{IND\text{-}CCA}^{\mathcal{A}}_{b,C_{clean}}$ is bounded by $\epsilon(\lambda)$.

DEFINITION 2.9. *(C_{clean}-FORGE Security [16]). Given a cleanness predicate C_{clean}, consider the $\mathsf{FORGE}^{\mathcal{A}}_{C_{\mathsf{clean}}}$ game in Fig. 3. We say that* SARCAD *is (q, ϵ)-C_{clean}-FORGE-secure if for any PPT adversary $\mathcal{A}$ bounded by $q(\lambda)$ queries to oracles, the advantage $Pr\left[\mathsf{FORGE}^{\mathcal{A}}_{C_{\mathsf{clean}}}(1^\lambda) \rightarrow 1\right]$ of $\mathcal{A}$ in $\mathsf{FORGE}^{\mathcal{A}}_{C_{\mathsf{clean}}}$ is bounded by $\epsilon(\lambda)$.*

To formalize IND-CCA-security, we use the following cleanness predicate C_{sym} from [16], defined in Fig. 4. Note that when there is a state exposure, the adversary can simulate all subsequent $\mathsf{RATCH}(P, \text{'rec'}, \cdot, \cdot)$ calls and state exposures.

$\mathsf{C}_{\mathsf{noexp}}$: for every P in $\{\mathsf{A},\mathsf{B}\}$, and every $\mathsf{EXP}_{\mathsf{st}}(\mathsf{P})$ call, P must have been the subject of a previous call either of the form $\mathsf{RATCH}(\mathsf{P},\text{“send”},\mathsf{ad}_{\mathsf{test}},\cdot)\rightarrow\mathsf{ct}_{\mathsf{test}}$ or $\mathsf{RATCH}(\mathsf{P},\text{“rec”},\mathsf{ad}_{\mathsf{test}},\mathsf{ct}_{\mathsf{test}})\rightarrow\mathsf{true}$.
$\mathsf{C}_{\mathsf{sym}}$: all the following conditions are satisfied:

- there is no $\mathsf{EXP}_{\mathsf{pt}}(\mathsf{P}_{\mathsf{test}})$ after time $\mathsf{t}_{\mathsf{test}}$ until there is a $\mathsf{RATCH}(\mathsf{P}_{\mathsf{test}},.,.,.)$ call;
- if the $\mathsf{CHALLENGE}$ 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 when $\overline{\mathsf{P}}_{\mathsf{test}}$ is in a matching status (Def. 2.6) at some time $\overline{\mathsf{t}}$, then there is no $\mathsf{EXP}_{\mathsf{pt}}(\overline{\mathsf{P}}_{\mathsf{test}})$ after time $\overline{\mathsf{t}}$ until there is another $\mathsf{RATCH}(\overline{\mathsf{P}}_{\mathsf{test}},.,.,.)$ call;
- ($\mathsf{C}_{\mathsf{noexp}}$) for every P in $\{\mathsf{A},\mathsf{B}\}$, and every $\mathsf{EXP}_{\mathsf{st}}(\mathsf{P})$ call, P must have been the subject of a previous call either of the form $\mathsf{RATCH}(\mathsf{P},\text{“send”},\mathsf{ad}_{\mathsf{test}},\cdot)\rightarrow\mathsf{ct}_{\mathsf{test}}$ or $\mathsf{RATCH}(\mathsf{P},\text{“rec”},\mathsf{ad}_{\mathsf{test}},\mathsf{ct}_{\mathsf{test}})\rightarrow\mathsf{true}$.

FIGURE 4. Cleanness predicates C_{sym} and C_{noexp}.

$\mathsf{Setup}(1^\lambda)$
1: $\mathsf{hk}\xleftarrow{\$}\mathsf{H}.\mathcal{K}(\lambda)$
2: **return** (λ,hk)

$\mathsf{Initall}(\mathsf{pp}=(\lambda,\mathsf{hk}))$
1: $\mathsf{k},\mathsf{k}'\xleftarrow{\$}\mathsf{OTAE}.\mathcal{K}(\lambda)$
2: $\mathsf{st}_A\leftarrow(\mathsf{hk},\mathsf{k},\mathsf{k}')$
3: $\mathsf{st}_B\leftarrow(\mathsf{hk},\mathsf{k}',\mathsf{k})$
4: **return** $(\mathsf{st}_A,\mathsf{st}_B)$

$\mathsf{Send}(\mathsf{st},\mathsf{ad},\mathsf{pt})$
1: parse $\mathsf{st}=(\mathsf{hk},\mathsf{sk},\mathsf{rk})$
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: $\mathsf{st}'\leftarrow(\mathsf{hk},\mathsf{sk}',\mathsf{rk})$
5: **return** $(\mathsf{st}',\mathsf{ct})$

$\mathsf{Receive}(\mathsf{st},\mathsf{ad},\mathsf{ct})$
1: parse $\mathsf{st}=(\mathsf{hk},\mathsf{sk},\mathsf{rk})$
2: $\mathsf{pt}\leftarrow\mathsf{OTAE.Dec}(\mathsf{rk},\mathsf{ad},\mathsf{ct})$
3: **if** $\mathsf{pt}=\perp$ **then**
4: **return** $(\mathsf{false},\mathsf{st},\perp)$
5: $\mathsf{rk}'\leftarrow\mathsf{H.Eval}(\mathsf{hk},\mathsf{rk})$
6: $\mathsf{st}'\leftarrow(\mathsf{hk},\mathsf{sk},\mathsf{rk}')$
7: **return** $(\mathsf{true},\mathsf{st}',\mathsf{pt})$

FIGURE 5. EtH: a SARCAD scheme.

Therefore, there is no possible healing after a state exposure. Thus, we prune out post-compromise security but leave forward secrecy. Note that our forward secrecy requirements automatically capture replay attacks and so they do not need to be explicitly considered. Note also that the IND-CCA game allows the adversary to make a single CHALLENGE query. By modifying the predicate C_{sym} and the IND-CCA game, it is possible to allow for multiple challenge queries, possibly from both parties, at the cost of a more complex definition and security analysis.

Similarly, we formalize FORGE-security by using the cleanness predicate C_{noexp}, defined in Fig. 4. In the FORGE game, performing a forgery becomes trivial when any $\mathsf{EXP}_{\mathsf{st}}(\cdot)$ occurs, due to the use of symmetric cryptography. Hence, no state exposure is allowed. Since there is no $(\mathsf{ad},\mathsf{ct})_{\mathsf{test}}$ message in FORGE-security, C_{noexp} here means no $\mathsf{EXP}_{\mathsf{st}}(\cdot)$ at all.

3. A SARCAD SCHEME BASED ON ENCRYPT-THEN-HASH

In this section, we propose our first SARCAD scheme, which we call EtH. Our scheme guarantees the same security properties as those proven for liteARCAD [16] and is a strictly superior alternative due to its increased efficiency.

3.1. Construction of EtH

Our construction EtH is depicted in Fig. 5. We call it *Encrypt-then-Hash* as we encrypt each message with an OTAE, after which the secret key is updated with hash function H. Here, we assume $\mathsf{OTAE}.\mathcal{K}(\lambda)=H.\mathcal{D}(\lambda)$.

In EtH, the state $\mathsf{st}_P=(\mathsf{hk},\mathsf{sk},\mathsf{rk})$ of a participant P includes three keys: a hash key, a sending key and a receiving key. Initially, the participants share two secret symmetric keys k and k', 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 OTAE scheme. Moreover, the key states are updated after each communication via a hash function. An example of a flow of messages is depicted in Fig. 6.

THEOREM 3.1. *(Correctness). Suppose that* OTAE *is correct. Then* EtH *is a correct* SARCAD.

Proof. In the EtH protocol, the two directions of communication are independent except that hk is used to hash in both directions. Therefore, the correctness of both directions will separately imply the correctness of the whole protocol.

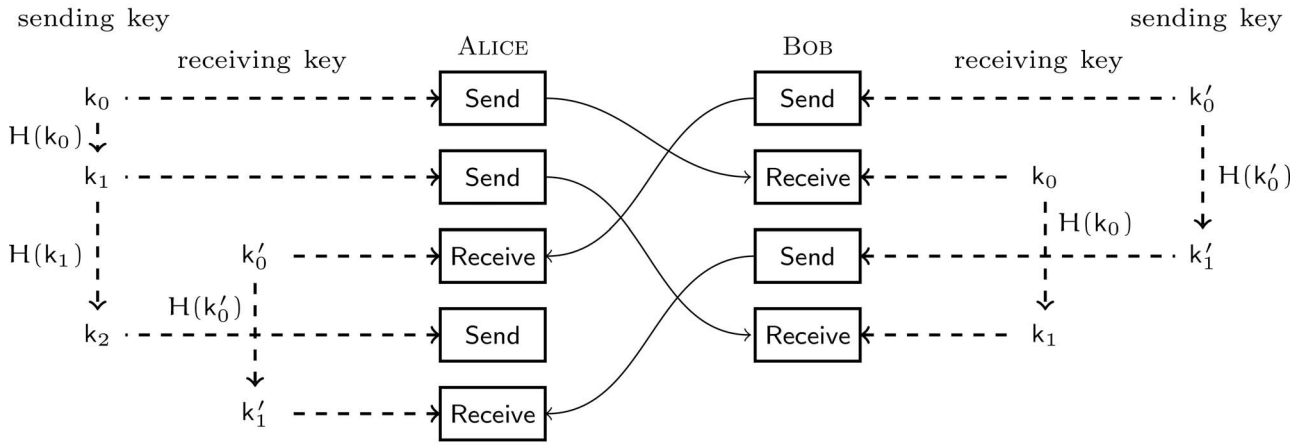


FIGURE 6. An example message exchange between Alice and Bob. At initialization, Alice and Bob share a sending key and receiving key $(\mathsf{k}_0, \mathsf{k}'_0)$ and $(\mathsf{k}'_0, \mathsf{k}_0)$, respectively. For the communication from Alice to Bob, Alice sends a message which is encrypted with k_0 and updates the sending key to k_1 by hashing k_0. Bob receives the message and decrypts it with k_0 and updates the receiving key to k_1 by hashing k_0 and so on. The process is analogous for the communication from Bob to Alice.

The correctness of the unidirectional case is trivial, which can be easily deduced from the correctness of the OTAE scheme and the fact that H.Eval is deterministic. 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 $\mathsf{send}^S_{\mathsf{pt}} = (\mathsf{seq}_1, (\mathsf{ad}, \mathsf{pt}), \mathsf{seq}_2)$ and $\mathsf{received}^R_{\mathsf{pt}} = \mathsf{seq}_1$ at the $(i\text{-}1)$-th step and let the key used for encrypting/decrypting the last message in seq_1 be k'. Now consider a $\mathsf{RATCH}(R, \text{`rec}', \cdot, \cdot)$ call at the $i$-th step of the loop, namely $\mathsf{sched}_i = (R, \text{`rec}')$. Let $\mathsf{received}^R_{\mathsf{pt}} = (\mathsf{seq}_1, (\mathsf{ad}', \mathsf{pt}'))$. Note that the key states are updated after each communication by H. Since H.Eval is deterministic, 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 $\mathsf{H.Eval}(\mathsf{hk}, \mathsf{k}')$. Further, due to the correctness of the OTAE scheme, we must have $(\mathsf{ad}', \mathsf{pt}') = (\mathsf{ad}, \mathsf{pt})$ and $\mathsf{RATCH}(R, \text{`rec}', \mathsf{ad}, \mathsf{ct}) = \mathsf{true}$. ■

The EtH scheme uses a hash function family with an *ad hoc PR* property (as is formally defined in Definition 3.1), which allows us to prove forward secrecy without resorting to the random oracle model. In practice, we can use e.g. AES-GCM [23] and SHA-256 [24] truncated to the required length. However, depending on performance requirements, using a hash function may be unnecessary as we can build a SARCAD using just an AEAD. We elaborate on this topic in Section 5.

DEFINITION 3.1. *(PR-Security). Let H be a hash function family and* OTAE *be a one-time authenticated encryption scheme such that* $\mathsf{OTAE}.\mathcal{K}(\lambda) = H.\mathcal{D}(\lambda)$. *We say that H is (q, ϵ)-PR for* OTAE *if for any PPT adversary $\mathcal{A}$ bounded by $q(\lambda)$ queries to oracles, the advantage*

$$Adv(\mathcal{A}) = Pr\left[\mathsf{PR}_0^{\mathcal{A}}(1^\lambda) \to 1\right] - Pr\left[\mathsf{PR}_1^{\mathcal{A}}(1^\lambda) \to 1\right]$$

of $\mathcal{A}$ in the $\mathsf{PR}_b^{\mathcal{A}}$ game is bounded by $\epsilon(\lambda)$.

Game $\mathsf{PR}_b^{\mathcal{A}}(1^\lambda)$:
1: $\mathsf{query}_{\mathsf{enc}} \leftarrow \perp$
2: $\mathsf{hk} \xleftarrow{\$} H.\mathcal{K}(\lambda)$
3: $\mathsf{k} \xleftarrow{\$} \mathsf{OTAE}.\mathcal{K}(\lambda)$
4: $\mathsf{k}'_0 \leftarrow H.\mathsf{Eval}(\mathsf{hk}, \mathsf{k})$
5: $\mathsf{k}'_1 \xleftarrow{\$} \mathsf{OTAE}.\mathcal{K}(\lambda)$
6: $\mathcal{A}^{\mathsf{ENC},\mathsf{DEC}}(\mathsf{hk}, \mathsf{k}'_b) \to z$
7: **return** z

Oracle $\mathsf{ENC}(\mathsf{ad}, \mathsf{pt})$
1: **if** $\mathsf{query}_{\mathsf{enc}} \neq \perp$ **then**
2: abort
3: **end if**
4: $\mathsf{ct} \leftarrow \mathsf{OTAE.Enc}(\mathsf{k}, \mathsf{ad}, \mathsf{pt})$
5: $\mathsf{query}_{\mathsf{enc}} \leftarrow \mathsf{ct}$
6: **return** ct

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

THEOREM 3.2. *Consider the* SARCAD *scheme* EtH *in Fig. 5. If* OTAE *is (q, ϵ_2)-IND-OTCCA-secure and (q, ϵ_3)-SEF-OTCMA-secure, and H is (q, ϵ_1)-*PR *secure for* OTAE, *then the scheme is (q, ϵ_4)-C_{sym}-IND-CCA-secure and (q, ϵ_5)-C_{noexp}-FORGE-secure, where $\epsilon_4 = q \cdot (q+1)\epsilon_1 + q \cdot \epsilon_2$ and $\epsilon_5 = q \cdot (q+1)\epsilon_1 + 2q^2 \cdot \epsilon_3$.*

Proof. The complete proofs of FORGE-security and IND-CCA-security are provided in Sections 3.2 and 3.3, respectively. ■

Theorems 3.1 and 3.2 show that EtH satisfies the conditions of results from Caforio *et al.* [16], which allow it to play the role of a weakly secure ARCAD in their hybrid 'on-demand

Game $\Gamma_{Q,m,n}(1^\lambda)$

1: $hk \xleftarrow{\$} H.\mathcal{K}(\lambda)$
2: $cnt^{send}_Q, cnt^{rec}_{\overline{Q}} \leftarrow 0$
3: pick $k_1, \ldots, k_m, k \xleftarrow{\$} OTAE.\mathcal{K}(\lambda)$
4: $st_Q \leftarrow (hk, k_1, k)$
5: $st_{\overline{Q}} \leftarrow (hk, k, k_1)$
6: set all $sent^*_*$ and $received^*_*$ variables to $\emptyset$
7: $(P, ad^*, ct^*) \leftarrow \mathcal{A}^{RATCH, EXP_{st}, EXP_{pt}}(hk)$
8: **if** $P \neq \overline{Q}$ or $cnt^{rec}_{\overline{Q}} \neq n-1$ **then** abort
9: RATCH(P,"rec",ad*,ct*)→ acc
10: **if** $\neg C_{clean}$ **then return** 0
11: **if** acc = false **then return** 0
12: **if** (ad^*, ct^*) is not a forgery for P **then return** 0
13: **return** 1

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

1: $pt_P \leftarrow pt$
2: $ad_P \leftarrow ad$
3: parse $st_P = (hk, sk, rk)$
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: $\quad\quad i \leftarrow cnt^{send}_Q + 1$
10: $\quad\quad sk' \leftarrow k_i$
11: $st_P \leftarrow (hk, sk', rk)$
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 $st_P = (hk, sk, rk)$
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: $\quad cnt^{rec}_{\overline{Q}} \leftarrow cnt^{rec}_{\overline{Q}} + 1$
10: $\quad$ **if** $cnt^{rec}_{\overline{Q}} \leqslant m-1$ **then**
11: $\quad\quad i \leftarrow cnt^{rec}_{\overline{Q}} + 1$
12: $\quad\quad rk' \leftarrow k_i$
13: $st_{\overline{P}} \leftarrow (hk, sk, rk')$
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

FIGURE 7. FORGE-security: hybrids $\Gamma_{Q,m,n}$.

ratcheting' construction. EtH can also be generically strengthened using techniques from Caforio *et al.* [16] to provide so-called security awareness. For instance, if an adversary performs a trivial forgery after a state exposure, the participants can notice it by seeing they can no longer communicate. Furthermore, when a participant P receives a message pt, they can deduce which messages were received by $\overline{P}$ when $\overline{P}$ sent pt given $\overline{P}$ is honest.

Note that we do not expect full security (especially post-compromise security) from this symmetric-only protocol. As pointed out by DV in [18], a secure and correct unidirectional ARCAD, which allows clean sender state exposures implies public-key encryption.

3.2. FORGE-security

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

We define below for each (Q, m, n) the hybrids $\Gamma_{Q,m,n}$ (refer to Fig. 7), which essentially assumes that the total number of accepting $RATCH(\overline{Q}, \text{'rec'}, \cdot, \cdot)$ calls by the adversary is at least $n-1$ i.e. a forgery attempt occurs when $\overline{Q}$ attempts to receive its nth message. The game maintains two counters: one counter cnt^{send}_Q for the number of messages sent by Q and one counter $cnt^{rec}_{\overline{Q}}$ for the number of messages received and accepted by $\overline{Q}$. For the unidirectional communication from Q to $\overline{Q}$, the m first session keys are picked randomly, whereas the following session keys are just updated by hashing the previous one as in

EtH. This is implemented by (i) preparing m randomly chosen keys $\mathsf{k}_1, \ldots, \mathsf{k}_m$ in the Initall phase and (ii) modifying the RATCH(Q, 'send', $\cdot$, $\cdot$) oracle (Line 6-10) and the RATCH($\overline{Q}$, 'rec', $\cdot$, $\cdot$) oracle (Line 8-12). Finally, we have that

$$\Pr[\Gamma \to 1] = \sum_{Q,n} \Pr\left[\Gamma_{Q,1,n} \to 1\right]. \quad (1)$$

In the following, we will prove that for $1 \leq m \leq n$, the difference between $\Pr\left[\Gamma_{Q,m,n} \to 1\right]$ and $\Pr\left[\Gamma_{Q,m+1,n} \to 1\right]$ is bounded. Recall that the $(m+1)$-th session key k_{m+1} in $\Gamma_{Q,m,n}$ is generated by hashing k_m while in $\Gamma_{Q,m+1,n}$ it is picked at random. This is the only difference. For any distinguisher $\mathcal{A}$ playing game which is either $\Gamma_{Q,m,n}$ or $\Gamma_{Q,m+1,n}$, define an adversary $\mathcal{B}_1$ (refer to Fig. 8) playing the following game $\mathsf{PR}_{b'}^{\mathcal{B}_1}$:

Game $\mathsf{PR}_{b'}^{\mathcal{B}_1}(1^\lambda)$:

1: $\mathsf{query}_{\mathsf{enc}} \leftarrow \perp$
2: $\mathsf{hk} \xleftarrow{\$} H.\mathcal{K}(\lambda)$
3: $\mathsf{k}_m \xleftarrow{\$} \mathsf{OTAE}.\mathcal{K}(\lambda)$
4: $\mathsf{k}' \leftarrow H.\mathsf{Eval}(\mathsf{hk}, \mathsf{k}_m)$
5: **if** $b' = 0$ **then** replace k' by a random value with the same length
6: **end if**
7: $\mathcal{B}_1^{\mathsf{ENC},\mathsf{DEC}}(\mathsf{hk}, \mathsf{k}') \to z$
8: **return** z

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

1: **if** $\mathsf{query}_{\mathsf{enc}} \neq \perp$ **then** abort
2: **end if**
3: $\mathsf{ct} \leftarrow \mathsf{OTAE.Enc}(\mathsf{k_m}, \mathsf{ad}, \mathsf{pt})$
4: $\mathsf{query}_{\mathsf{enc}} \leftarrow \mathsf{ct}$
5: **return** ct

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

1: $\mathsf{pt} \leftarrow \mathsf{OTAE.Dec}(\mathsf{k_m}, \mathsf{ad}, \mathsf{ct})$
2: **return** pt

The adversary $\mathcal{B}_1$ can simulate the difference between $\Gamma_{Q,m,n}$ and $\Gamma_{Q,m+1,n}$ by using k' he received in game $\mathsf{PR}_{b'}^{\mathcal{B}_1}$, which is either a hash value or a random value, as the $(m+1)$th session key k_{m+1}. We can see that the advantage of $\mathcal{B}_1$ is

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

which is bounded by ϵ_1 due to PR-security (recall Definition 3.1).

Note that $\left|\Pr\left[\Gamma_{Q,1,n} \to 1\right] - \Pr\left[\Gamma_{Q,n+1,n} \to 1\right]\right| \leq \sum_{m=1}^{n} \left|\Pr\left[\Gamma_{Q,m,n} \to 1\right] - \Pr\left[\Gamma_{Q,m+1,n} \to 1\right]\right|$, thereby bounded by $n \cdot \epsilon_1$, and since $1 \leq m \leq n \leq q$, we have $q \cdot (q+1)$ hybrids in total. Combined with Equation (1), we can then deduce that

$$\left|\Pr[\Gamma \to 1] - \sum_{Q,n} \Pr\left[\Gamma_{Q,n+1,n} \to 1\right]\right| \leq q \cdot (q+1)\epsilon_1. \quad (2)$$

For any distinguisher $\mathcal{A}$ playing game $\Gamma_{Q,n+1,n}$, we define an adversary $\mathcal{E}_x$ (refer to Fig. 9) playing the following game SEF-OTCMA with respect to integer $1 \leq x \leq q$:

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

1: $\mathsf{k}_n \xleftarrow{\$} \mathsf{OTAE}.\mathcal{K}(\lambda)$
2: $\mathsf{query}_{\mathsf{enc}} \leftarrow \perp$
3: $\mathcal{E}^{\mathsf{ENC}}(1^\lambda) \to (\mathsf{ad}^*, \mathsf{ct}^*)$
4: **if** $(\mathsf{ad}^*, \mathsf{ct}^*) = \mathsf{query}_{\mathsf{enc}}$ **then** abort
5: **end if**
6: **if** $\mathsf{OTAE.Dec}(\mathsf{k}_n, \mathsf{ad}^*, \mathsf{ct}^*) = \perp$ **then** abort
7: **end if**
8: **return** 1

Meanwhile, $\mathcal{E}_x$ makes a forgery by using the forgery given by $\mathcal{A}$ in game $\Gamma_{Q,n+1,n}$. We abort at Line 7 of RATCH(P, 'rec', ad, ct) after x decryptions to simulate decryption failures. Note that we can safely abort in the case of successful decryption after x decryptions since we would have $\mathsf{cnt}_{\overline{Q}}^{\mathsf{rec}} \neq n-1$ and thus $\mathcal{E}$ would abort anyway. Moreover, there exist at most q adversaries of the form $\mathcal{E}_{x'}$ for appropriate x' that can perfectly simulate $\Gamma_{Q,n+1,n}$ together. Thus, the advantage of $\mathcal{A}$, which is

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

is upper bounded at most q times the advantage of $\mathcal{E}_x$, namely $q \cdot \epsilon_3$, by applying the union bound. Finally, we deduce that the probability $\Pr[\Gamma \to 1]$ is bounded by $q \cdot (q+1) \cdot \epsilon_1 + 2q^2 \cdot \epsilon_3$ by combining this with Equation (2).

3.3. IND-CCA-security

We then consider IND-CCA-security. We define Γ_b as the initial C_{sym}-IND-CCA game which has a challenge message $(\mathsf{ad}_{\mathsf{test}}, \mathsf{pt}_{\mathsf{test}})$ returning $\mathsf{ct}_{\mathsf{test}}$. The game Γ_b has the property that whenever C_{sym} does not occur, it never returns a bit z due to the C_{sym} cleanness condition.

Similarl to the FORGE-security proof, we define below for each (Q, m, n) the hybrids $\Gamma_{Q,m,n}^b$ (refer to Fig. 10), which essentially assumes that the challenge message is the nth message sent by Q. 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 CHALLENGE oracle (Line 5-6). Clearly, we have that

$$\Pr[\Gamma_b \to 1] = \sum_{Q,n} \Pr\left[\Gamma_{Q,1,n}^b \to 1\right]. \quad (3)$$

Moreover, we have that

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

$\mathcal{B}_1^{\mathsf{ENC},\mathsf{DEC}}(\mathsf{hk},\mathsf{k}')$:

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

Subroutine $\mathsf{RATCH}(\mathsf{P},$ “send”, $\mathsf{ad}, \mathsf{pt})$

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

Subroutine $\mathsf{RATCH}(\mathsf{P},$ “rec”, $\mathsf{ad}, \mathsf{ct})$

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

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

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

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

1: **return** pt_{P}

FIGURE 8. Adversary $\mathcal{B}_1$ against PR security based on an adversary distinguishing between $\Gamma_{Q,m,n}$ and $\Gamma_{Q,m+1,n}$.

is bounded by ϵ_1 due to PR-security (Fig. 8 reduction). Therefore, we have

$$\begin{aligned}
&\left|\Pr\left[\Gamma^b_{Q,1,n} \rightarrow 1\right] - \Pr\left[\Gamma^b_{Q,n+1,n} \rightarrow 1\right]\right| \\
&\leq \sum_{m=1}^{n} \left|\Pr\left[\Gamma^b_{Q,m,n} \rightarrow 1\right] - \Pr\left[\Gamma^b_{Q,m+1,n} \rightarrow 1\right]\right| \\
&\leq n \cdot \epsilon_1.
\end{aligned}$$

Combined with Equation (3), we can then deduce that

$$\left|\Pr\left[\Gamma_b \rightarrow 1\right] - \sum_{Q,n} \Pr\left[\Gamma^b_{Q,n+1,n} \rightarrow 1\right]\right| \leq q \cdot (q+1)\epsilon_1. \quad (4)$$

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 C_{sym}-cleanness, no participant has an $\mathsf{EXP}_{\mathsf{st}}$ before

$\mathcal{E}_x^{\mathsf{ENC}}(1^\lambda)$:
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 $\mathsf{k}_1, \ldots, \mathsf{k}_{n-1}, \mathsf{k}_{n+1}, \mathsf{k} \xleftarrow{\$} \mathsf{OTAE}.\mathcal{K}(\lambda)$, $\mathsf{k}_n \leftarrow \perp$
4: $\mathsf{st}_{\mathrm{Q}} \leftarrow (\mathsf{hk}, \mathsf{k}_1, \mathsf{k})$
5: $\mathsf{st}_{\overline{\mathrm{Q}}} \leftarrow (\mathsf{hk}, \mathsf{k}, \mathsf{k}_1)$
6: set all sent_*^* and $\mathsf{received}_*^*$ variables to $\emptyset$
7: $\mathsf{dec\text{-}cnt} \leftarrow 0$
8: $(\mathrm{P}, \mathsf{ad}^*, \mathsf{ct}^*) \leftarrow \mathcal{A}^{\mathsf{RATCH},\mathsf{EXP}_{\mathsf{st}},\mathsf{EXP}_{\mathsf{pt}}}(\mathsf{hk})$
9: **if** $\mathrm{P} \neq \overline{\mathrm{Q}}$ or $\mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} \neq n-1$ **then** abort
10: **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 $\mathsf{st}_{\mathrm{P}} = (\mathsf{hk}, \mathsf{sk}, \mathsf{rk})$
4: $\mathsf{sk}' \leftarrow \mathrm{H}.\mathsf{Eval}(\mathsf{hk}, \mathsf{sk})$
5: **if** $\mathrm{P} = \mathrm{Q}$ and $\mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} = n-1$ **then**
6: $\quad \mathsf{ct}_{\mathrm{P}} \leftarrow \mathsf{ENC}(\mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$
7: $\quad \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} \leftarrow \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} + 1$
8: $\quad \mathsf{sk}' \leftarrow \mathsf{k}_{n+1}$
9: **else**
10: $\quad \mathsf{ct}_{\mathrm{P}} \leftarrow \mathsf{OTAE.Enc}(\mathsf{sk}, \mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$
11: $\quad$ **if** $\mathrm{P} = \mathrm{Q}$ **then**
12: $\quad\quad \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} \leftarrow \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} + 1$
13: $\quad\quad$ **if** $\mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} \leqslant n-1$ **then**
14: $\quad\quad\quad \mathsf{i} \leftarrow \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} + 1$
15: $\quad\quad\quad \mathsf{sk}' \leftarrow \mathsf{k}_{\mathsf{i}}$
16: $\mathsf{st}_{\mathrm{P}} \leftarrow (\mathsf{hk}, \mathsf{sk}', \mathsf{rk})$
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 $\mathsf{st}_{\mathrm{P}} = (\mathsf{hk}, \mathsf{sk}, \mathsf{rk})$
4: $\mathsf{rk}' \leftarrow \mathrm{H}.\mathsf{Eval}(\mathsf{hk}, \mathsf{rk})$
5: **if** $\mathrm{P} = \overline{\mathrm{Q}}$ and $\mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} = n-1$ **then**
6: $\quad \mathsf{dec\text{-}cnt} \leftarrow \mathsf{dec\text{-}cnt} + 1$
7: $\quad$ **if** $\mathsf{dec\text{-}cnt} = x$ **then**
8: $\quad\quad$ abort
9: $\quad$ **else return** false
10: **else**
11: $\quad \mathsf{pt}_{\mathrm{P}} \leftarrow \mathsf{OTAE.Dec}(\mathsf{rk}, \mathsf{ad}, \mathsf{ct})$
12: $\quad$ **if** $\mathsf{pt}_{\mathrm{P}} = \perp$ **then**
13: $\quad\quad$ **return** false
14: $\quad$ **if** $\mathrm{P} = \overline{\mathrm{Q}}$ **then**
15: $\quad\quad \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} \leftarrow \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} + 1$
16: $\quad\quad$ **if** $\mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} \leqslant n-1$ **then**
17: $\quad\quad\quad \mathsf{i} \leftarrow \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} + 1$
18: $\quad\quad\quad \mathsf{rk}' \leftarrow \mathsf{k}_{\mathsf{i}}$
19: $\mathsf{st}_{\overline{\mathrm{P}}} \leftarrow (\mathsf{hk}, \mathsf{sk}, \mathsf{rk}')$
20: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$ to $\mathsf{received}_{\mathsf{pt}}^{\mathrm{P}}$
21: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{ct}_{\mathrm{P}})$ to $\mathsf{received}_{\mathsf{ct}}^{\mathrm{P}}$
22: **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}

FIGURE 9. Adversary $\mathcal{E}_x$ against $\mathsf{OTAESEF} - \mathsf{OTCMA}$-security based on a $\Gamma_{Q,n+1,n}$ adversary $\mathcal{A}$ (FORGE security).

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 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 bounded by ϵ_2 by the $\mathsf{IND\text{-}OTCCA}$ 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. 11) playing the following game $\mathsf{IND\text{-}OTCCA}_b^{\mathcal{D}}$:

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

Oracle $\mathsf{DEC}(\mathsf{ad}, \mathsf{ct})$
1: **if** $(\mathsf{ad}, \mathsf{ct}) =$ challenge **then** abort
2: **end if**
3: **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: **end if**
3: **if** $\mathsf{b} = 0$ **then**
4: $\quad$ replace pt by a random message of the same length
5: **end if**
6: $\mathsf{OTAE.Enc}(k_n, \mathsf{ad}, \mathsf{pt}) \to \mathsf{ct}$

Game $\Gamma^{b}_{Q,m,n}(1^\lambda)$:

$\mathsf{hk} \xleftarrow{\$} \mathrm{H}.\mathcal{K}(\lambda)$
$\mathsf{cnt}^{\mathsf{send}}_{\mathrm{Q}}, \mathsf{cnt}^{\mathsf{rec}}_{\overline{\mathrm{Q}}} \leftarrow 0$
pick $\mathsf{k}_1, \ldots, \mathsf{k}_m, \mathsf{k} \xleftarrow{\$} \mathrm{OTAE}.\mathcal{K}(\lambda)$
$\mathsf{st}_{\mathrm{Q}} \leftarrow (\mathsf{hk}, \mathsf{k}_1, \mathsf{k})$
$\mathsf{st}_{\overline{\mathrm{Q}}} \leftarrow (\mathsf{hk}, \mathsf{k}, \mathsf{k}_1)$
set sent^*_* and $\mathsf{received}^*_*$ variables to $\emptyset$
set $\mathsf{t}_{\mathsf{test}}$ to $\perp$
$z \leftarrow \mathcal{A}^{\mathsf{RATCH},\mathsf{EXP}_{\mathsf{st}},\mathsf{EXP}_{\mathsf{pt}},\mathsf{CHALLENGE}}(\mathsf{hk})$
if $\neg \mathsf{C}_{\mathsf{clean}}$ **then return** $\perp$
return z

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

$\mathsf{pt}_{\mathrm{P}} \leftarrow \mathsf{pt}$
$\mathsf{ad}_{\mathrm{P}} \leftarrow \mathsf{ad}$
parse $\mathsf{st}_{\mathrm{P}} = (\mathsf{hk}, \mathsf{sk}, \mathsf{rk})$
$\mathsf{ct}_{\mathrm{P}} \leftarrow \mathrm{OTAE.Enc}(\mathsf{sk}, \mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$
$\mathsf{sk}' \leftarrow \mathrm{H.Eval}(\mathsf{hk}, \mathsf{sk})$
if $\mathrm{P} = \mathrm{Q}$ **then**
  $\mathsf{cnt}^{\mathsf{send}}_{\mathrm{Q}} \leftarrow \mathsf{cnt}^{\mathsf{send}}_{\mathrm{Q}} + 1$
  if $\mathsf{cnt}^{\mathsf{send}}_{\mathrm{Q}} \leqslant m - 1$ **then**
    $\mathsf{i} \leftarrow \mathsf{cnt}^{\mathsf{send}}_{\mathrm{Q}} + 1$
    $\mathsf{sk}' \leftarrow \mathsf{k}_{\mathsf{i}}$
$\mathsf{st}_{\mathrm{P}} \leftarrow (\mathsf{hk}, \mathsf{sk}', \mathsf{rk})$
append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$ to $\mathsf{sent}^{\mathrm{P}}_{\mathsf{pt}}$
append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{ct}_{\mathrm{P}})$ to $\mathsf{sent}^{\mathrm{P}}_{\mathsf{ct}}$
return ct_{P}

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

$\mathsf{ct}_{\mathrm{P}} \leftarrow \mathsf{ct}$
$\mathsf{ad}_{\mathrm{P}} \leftarrow \mathsf{ad}$
parse $\mathsf{st}_{\mathrm{P}} = (\mathsf{hk}, \mathsf{sk}, \mathsf{rk})$
$\mathsf{pt}_{\mathrm{P}} \leftarrow \mathrm{OTAE.Dec}(\mathsf{rk}, \mathsf{ad}, \mathsf{ct})$
if $\mathsf{pt}_{\mathrm{P}} = \perp$ **then**
  return false
$\mathsf{rk}' \leftarrow \mathrm{H.Eval}(\mathsf{hk}, \mathsf{rk})$
if $\mathrm{P} = \overline{\mathrm{Q}}$ **then**
  $\mathsf{cnt}^{\mathsf{rec}} \leftarrow \mathsf{cnt}^{\mathsf{rec}}_{\overline{\mathrm{Q}}} + 1$
  if $\mathsf{cnt}^{\mathsf{rec}}_{\overline{\mathrm{Q}}} \leqslant m - 1$ **then**
    $\mathsf{i} \leftarrow \mathsf{cnt}^{\mathsf{rec}}_{\overline{\mathrm{Q}}} + 1$
    $\mathsf{rk}' \leftarrow \mathsf{k}_{\mathsf{i}}$
$\mathsf{st}_{\overline{\mathrm{P}}} \leftarrow (\mathsf{hk}, \mathsf{sk}, \mathsf{rk}')$
append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$ to $\mathsf{received}^{\mathrm{P}}_{\mathsf{pt}}$
append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{ct}_{\mathrm{P}})$ to $\mathsf{received}^{\mathrm{P}}_{\mathsf{ct}}$
return true

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

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
return st_{P}

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

return pt_{P}

Oracle CHALLENGE(P, ad, pt)

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

FIGURE 10. IND-CCA-security: hybrids $\Gamma^{b}_{Q,m,n}$.

challenge $\leftarrow$ (ad, ct)
$\Re$ turn ct

Adversary $\mathcal{D}$ playing $\mathsf{IND\text{-}OTCCA}_b$ with respect to bit b can simulate $\Gamma^{b}_{Q,n+1,n}$ 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. 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 bounded by ϵ_2 due to the IND-OTCCA security of OTAE.

Finally, we can deduce that the difference $|\Pr[\Gamma_0 \to 1] - \Pr[\Gamma_1 \to 1]|$ is bounded by $q \cdot (q+1)\epsilon_1 + q \cdot \epsilon_2$ by Equation (4).

4. A NEW SARCAD SCHEME BASED ON AN INTEGRATED PRIMITIVE

In this section, we propose another SARCAD scheme, which we call EtUK that is based on a newly introduced integrated OTEAE primitive.

4.1. An integrated primitive OTEAE

We extend the definition of OTAE to a primitive called a OTEAE scheme, which provides an additional PR output block that can be later used as a secret key.

$\mathcal{D}^{\mathsf{DEC},\mathsf{CH}}(1^\lambda)$:

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 $\mathsf{k}_1, \ldots, \mathsf{k}_{n-1}, \mathsf{k}_{n+1}, \mathsf{k} \xleftarrow{\$} \mathsf{OTAE}.\mathcal{K}(\lambda)$
4: set $\mathsf{k}_n \leftarrow \perp$
5: $\mathsf{st}_{\mathrm{Q}} \leftarrow (\mathsf{hk}, \mathsf{k}_1, \mathsf{k})$
6: $\mathsf{st}_{\overline{\mathrm{Q}}} \leftarrow (\mathsf{hk}, \mathsf{k}, \mathsf{k}_1)$
7: set sent_*^* and $\mathsf{received}_*^*$ variables to $\emptyset$
8: set $\mathsf{t}_{\mathsf{test}}$ to $\perp$
9: $\mathsf{b}' \leftarrow \mathcal{A}^{\mathsf{RATCH},\mathsf{EXP}_{\mathsf{st}},\mathsf{EXP}_{\mathsf{pt}},\mathsf{CHALLENGE}}(\mathsf{hk})$
10: **if** $\neg \mathsf{C}_{\mathsf{clean}}$ **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 $\mathsf{st}_{\mathrm{P}} = (\mathsf{hk}, \mathsf{sk}, \mathsf{rk})$
4: $\mathsf{ct}_{\mathrm{P}} \leftarrow \mathsf{OTAE.Enc}(\mathsf{sk}, \mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$
5: $\mathsf{sk}' \leftarrow \mathrm{H}.\mathsf{Eval}(\mathsf{hk}, \mathsf{sk})$
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: $\quad\quad \mathsf{i} \leftarrow \mathsf{cnt}_{\mathrm{Q}}^{\mathsf{send}} + 1$
14: $\quad\quad \mathsf{sk}' \leftarrow \mathsf{k}_{\mathsf{i}}$
15: $\mathsf{st}_{\mathrm{P}} \leftarrow (\mathsf{hk}, \mathsf{sk}', \mathsf{rk})$
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 $\mathsf{st}_{\mathrm{P}} = (\mathsf{hk}, \mathsf{sk}, \mathsf{rk})$
4: $\mathsf{rk}' \leftarrow \mathrm{H}.\mathsf{Eval}(\mathsf{hk}, \mathsf{rk})$
5: **if** $\mathrm{P} = \overline{\mathrm{Q}}$ and $\mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} = n - 1$ **then**
6: $\quad \mathsf{pt}_{\mathrm{P}} \leftarrow \perp$
7: $\quad$ **if** $\mathsf{ad} = \mathsf{ad}_{\mathrm{test}}$ and $\mathsf{ct} = \mathsf{ct}_{\mathrm{test}}$ **then**
8: $\quad\quad \mathsf{pt}_{\mathrm{P}} \leftarrow \mathsf{pt}_{\mathrm{test}}$
9: $\quad$ **else**
10: $\quad\quad \mathsf{pt}_{\mathrm{P}} \leftarrow \mathsf{DEC}(\mathsf{ad}, \mathsf{ct})$
11: $\quad$ **if** $\mathsf{pt}_{\mathrm{P}} = \perp$ **then**
12: $\quad\quad$ **return** false
13: $\quad$ **else**
14: $\quad\quad \mathsf{rk}' \leftarrow \mathsf{k}_{n+1}$
15: **else**
16: $\quad \mathsf{pt}_{\mathrm{P}} \leftarrow \mathsf{OTAE.Dec}(\mathsf{rk}, \mathsf{ad}, \mathsf{ct})$
17: $\quad$ **if** $\mathsf{pt}_{\mathrm{P}} = \perp$ **then**
18: $\quad\quad$ **return** false
19: $\quad$ **if** $\mathrm{P} = \overline{\mathrm{Q}}$ **then**
20: $\quad\quad \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} \leftarrow \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} + 1$
21: $\quad\quad$ **if** $\mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} \leqslant n - 1$ **then**
22: $\quad\quad\quad \mathsf{i} \leftarrow \mathsf{cnt}_{\overline{\mathrm{Q}}}^{\mathsf{rec}} + 1$
23: $\quad\quad\quad \mathsf{rk}' \leftarrow \mathsf{k}_{\mathsf{i}}$
24: $\mathsf{st}_{\mathrm{P}} \leftarrow (\mathsf{hk}, \mathsf{sk}, \mathsf{rk}')$
25: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{pt}_{\mathrm{P}})$ to $\mathsf{received}_{\mathsf{pt}}^{\mathrm{P}}$
26: append $(\mathsf{ad}_{\mathrm{P}}, \mathsf{ct}_{\mathrm{P}})$ to $\mathsf{received}_{\mathsf{ct}}^{\mathrm{P}}$
27: **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

FIGURE 11. Adversary $\mathcal{D}$ against $\mathsf{OTAEIND} - \mathsf{OTCCA}$-security based on a $\Gamma^b_{Q,n+1,n}$ adversary $\mathcal{A}$ ($\mathsf{IND\text{-}CCA}$-security).

DEFINITION 4.1. *(*OTEAE*). We define a OTEAE scheme with associated data, which consists of a key space* $\mathsf{OTEAE}.\mathcal{K}(\lambda)$ *and two polynomially bounded deterministic algorithms: (i)* $\mathsf{OTEAE.Enc}$ *takes a key* k *in* $\mathcal{K}(\lambda)$*, an associated data* $\mathsf{ad} \in \mathcal{A}(\lambda)$ *and a message* $\mathsf{pt} \in \mathcal{D}(\lambda)$ *and returns a ciphertext* $\mathsf{ct} \in \mathcal{C}(\lambda)$ *together with an updated new key* k' *in* $\mathsf{OTEAE}.\mathcal{K}(\lambda)$*,*

namely

$$(\mathsf{k}', \mathsf{ct}) = \mathsf{OTEAE.Enc}(\mathsf{k}, \mathsf{ad}, \mathsf{pt});$$

(ii) OTEAE.Dec *takes* k, ad *and* ct *to recover the message* pt *(else the distinguished symbol* $\perp$*) and update the key state to* k′*, namely*

$$(\mathsf{k}', \mathsf{pt}) = \mathsf{OTEAE.Dec}(\mathsf{k}, \mathsf{ad}, \mathsf{ct}).$$

Correctness. The correctness notion for an OTEAE scheme is that (i) standard correctness is satisfied with respect to encryption and decryption (i.e. encryption/decryption consistency) and (ii) both outputs $\mathsf{k}' \in \mathsf{OTEAE}.\mathcal{K}(\lambda)$ are equal. That is, for any key $\mathsf{k} \in \mathsf{OTEAE}.\mathcal{K}(\lambda)$, and for all associated data ad and messages pt, for $(\mathsf{k}_1, \mathsf{ct}) \leftarrow \mathsf{OTEAE.Enc}(\mathsf{k}, \mathsf{ad}, \mathsf{pt})$, $(\mathsf{k}_2, \mathsf{pt}') \leftarrow \mathsf{OTEAE.Dec}(\mathsf{k}, \mathsf{ad}, \mathsf{ct})$, it holds that: 1) $\mathsf{pt}' = \mathsf{pt}$; and 2) $\mathsf{k}_1 = \mathsf{k}_2$.

Security. We require the OTEAE scheme to satisfy the IND-OTCCA and SEF-OTCMA security notions. Moreover, we require that it additionally satisfies KIND security, which is related to key indistinguishability. Those security definitions are made so that Theorem 3.2 can easily be adapted to OTEAE, as stated in Section 4.2.

DEFINITION 4.2. *(*IND-OTCCA*-Security). An* OTEAE *scheme is said to be* (q, ϵ)*-*IND-OTCCA*-secure if for any PPT adversary* $\mathcal{A}$ *bounded by* $q(\lambda)$ *queires to oracles, the advantage*

$$\mathit{Adv}(\mathcal{A}) = Pr\left[\mathsf{IND-OTCCA}_0^{\mathcal{A}}(1^\lambda) \rightarrow 1\right] - Pr\left[\mathsf{IND-OTCCA}_1^{\mathcal{A}}(1^\lambda) \rightarrow 1\right]$$

of $\mathcal{A}$ *playing the following game is bounded by* $\epsilon(\lambda)$*.*

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

challenge $\leftarrow \perp$
$\mathsf{k} \xleftarrow{\$} \mathsf{OTEAE}.\mathcal{K}(\lambda)$
$\mathcal{A}^{\mathsf{CH},\mathsf{DEC}}(1^\lambda) \rightarrow b'$
return b'

Oracle CH(ad, pt)

if challenge $\neq \perp$ **then**
    abort
end if
if $\mathsf{b} = 0$ **then**
    replace pt by a random message of the same length
end if
$\mathsf{OTEAE.Enc}(\mathsf{k}, \mathsf{ad}, \mathsf{pt}) \rightarrow (\mathsf{k}', \mathsf{ct})$
challenge $\leftarrow (\mathsf{ad}, \mathsf{ct})$
return ct

Oracle DEC(ad, ct)

if $(\mathsf{ad}, \mathsf{ct}) =$ challenge **then** abort
end if
$\mathsf{OTEAE.Dec}(\mathsf{k}, \mathsf{ad}, \mathsf{ct}) \rightarrow (\mathsf{k}', \mathsf{pt})$
return pt

DEFINITION 4.3. *(*SEF-OTCMA*-Security). An* OTEAE *scheme is said to be* (q, ϵ)*-SEF-OTCMA-secure if for any PPT adversary* $\mathcal{A}$ *bounded by* $q(\lambda)$ *queries to oracles, the advantage of* $\mathcal{A}$ *playing the following game* $Pr\left[\mathsf{SEF-OTCMA}^{\mathcal{A}}(1^\lambda) \rightarrow 1\right]$ *is bounded by* $\epsilon(\lambda)$*.*

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

$\mathsf{k} \xleftarrow{\$} \mathsf{OTEAE}.\mathcal{K}(\lambda)$
$\mathsf{query}_{\mathsf{enc}} \leftarrow \perp$
$\mathcal{A}^{\mathsf{ENC}}(1^\lambda) \rightarrow (\mathsf{ad}', \mathsf{ct}')$
if $(\mathsf{ad}', \mathsf{ct}') = \mathsf{query}_{\mathsf{enc}}$ **then** abort
end if
if $\mathsf{OTEAE.Dec}(\mathsf{k}, \mathsf{ad}', \mathsf{ct}') = \perp$ **then** abort
end if
return 1

Oracle ENC(ad, pt)

if $\mathsf{query}_{\mathsf{enc}} \neq \perp$ **then** abort
end if
$\mathsf{OTEAE.Enc}(\mathsf{k}, \mathsf{ad}, \mathsf{pt}) \rightarrow (\mathsf{k}', \mathsf{ct})$
$\mathsf{query}_{\mathsf{enc}} \leftarrow (\mathsf{ad}, \mathsf{ct})$
return ct

DEFINITION 4.4. *(*KIND*-Security). An* OTEAE *scheme is said to be* (q, ϵ)*-*KIND*-secure if for any PPT adversary* $\mathcal{A}$ *bounded by* $q(\lambda)$ *queries to oracles, the advantage*

$$\mathit{Adv}(\mathcal{A}) = Pr\left[\mathsf{KIND}_0^{\mathcal{A}}(1^\lambda) \rightarrow 1\right] - Pr\left[\mathsf{KIND}_1^{\mathcal{A}}(1^\lambda) \rightarrow 1\right]$$

of $\mathcal{A}$ *playing the following game is bounded by* $\epsilon(\lambda)$*.*

Game $\mathsf{KIND}_b^{\mathcal{A}}(1^\lambda)$:

$\mathsf{query}_{\mathsf{enc}} \leftarrow \perp$
$\mathsf{k} \xleftarrow{\$} \mathsf{OTEAE}.\mathcal{K}(\lambda)$
$\mathcal{A}^{\mathsf{ENC},\mathsf{DEC}}(1^\lambda) \rightarrow z$
return z

Oracle ENC(ad, pt)

if $\mathsf{query}_{\mathsf{enc}} \neq \perp$ **then** abort
end if
$(\mathsf{k}', \mathsf{ct}) \leftarrow \mathsf{OTEAE.Enc}(\mathsf{k}, \mathsf{ad}, \mathsf{pt})$
if $b = 0$ **then** replace k′ by a random value in $\mathsf{OTEAE}.\mathcal{K}(\lambda)$
end if
$\mathsf{query}_{\mathsf{enc}} \leftarrow (\mathsf{ad}, \mathsf{pt})$
return $(\mathsf{k}', \mathsf{ct})$

Oracle DEC(ad, ct)

$(\mathsf{k}', \mathsf{pt}) \leftarrow \mathsf{OTEAE.Dec}(\mathsf{k}, \mathsf{ad}, \mathsf{ct})$
return pt

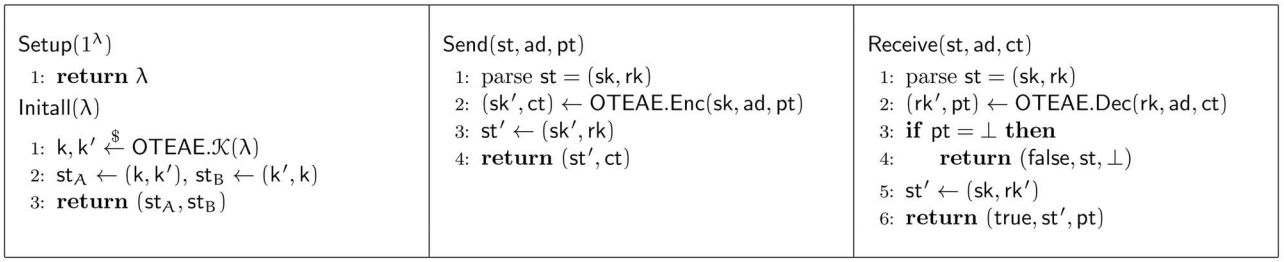


FIGURE 12. EtUK: A SARCAD Scheme Based on OTEAE.

4.2. A new SARCAD scheme

In the following, we give the new SARCAD scheme, as depicted in Fig. 12. We call it EtUK. In EtUK, the state $\mathsf{st}_P = (\mathsf{sk}, \mathsf{rk})$ of a participant P includes two keys, a sending key sk for encrypting (sending) messages and a receiving key rk for decrypting (receiving) messages, which are initialized as k (k′) and k′ (k), respectively.

Unlike the previous scheme EtH, which is instantiated with standard primitives (an OTAE and a hash function) with an *ad hoc* security notion between them introduced, the new scheme is based on the integrated primitive OTEAE.

Correctness of EtUK. The EtUK scheme is a correct SARCAD if OTEAE is correct. The proof is almost the same with that of EtH. Note that the correctness of the whole protocol is guaranteed by correctness in both directions, which are still independent, while the correctness of the unidirectional case can be trivially deduced from the correctness of the OTEAE scheme.

Security of EtUK. The EtUK scheme is a secure SARCAD if OTEAE is secure, which is formally given in the following theorem.

THEOREM 4.1. *Consider the* SARCAD *scheme* EtUK *in Fig. 12. If OTEAE is* (q, ϵ_1)*-IND-OTCCA-secure,* (q, ϵ_2)*-SEF-OTCMA-secure, as well as* (q, ϵ_3)*-*KIND*-secure, then the scheme is* $q \cdot \epsilon_1 + q \cdot (q+1)\epsilon_3$*-*$C_{sym}$*-IND-CCA-secure and* $2q^2 \cdot \epsilon_2 + q \cdot (q+1)\epsilon_3$*-*$C_{noexp}$*-FORGE-secure.*

Proof sketch. The security proof parallels that of EtH. We are not going to cover it from beginning to end because of the technical similarities. Note that the indistinguishability of hybrids now reduces to the KIND security of OTEAE. More specifically, we define as follows an adversary $\mathcal{B}_2$ (refer to Fig. 13) playing the following game $\mathsf{KIND}^{\mathcal{B}_2}_{b'}$, which can simulate the difference between hybrids $\Gamma_{Q,m,n}$ and $\Gamma_{Q,m+1,n}$.

Game $\mathsf{KIND}^{\mathcal{B}_2}_b(1^\lambda)$:
1: $\mathsf{query}_{\mathsf{enc}} \leftarrow \perp$
2: $\mathsf{k}_m \xleftarrow{\$} \mathsf{OTEAE}.\mathcal{K}(\lambda)$
3: $\mathcal{B}_2^{\mathsf{ENC},\mathsf{DEC}}(1^\lambda) \rightarrow z$
4: **return** z

Oracle ENC(ad, pt)
1: **if** $\mathsf{query}_{\mathsf{enc}} \neq \perp$ **then** abort
2: **end if**
3: $(\mathsf{k}', \mathsf{ct}) \leftarrow \mathsf{OTEAE.Enc}(\mathsf{k}_m, \mathsf{ad}, \mathsf{pt})$
4: **if** $b = 0$ **then** replace k′ by a random value with the same length
5: **end if**
6: $\mathsf{query}_{\mathsf{enc}} \leftarrow (\mathsf{ad}, \mathsf{pt})$
7: **return** $(\mathsf{k}', \mathsf{ct})$

Oracle DEC(ad, ct)
1: $(\mathsf{k}', \mathsf{pt}) \leftarrow \mathsf{OTEAE.Dec}(\mathsf{k}_m, \mathsf{ad}, \mathsf{ct})$
2: **return** pt

After this reduction, when considering FORGE-security, we can prove that

$$\Pr\left[\Gamma_{Q,n+1,n} \rightarrow 1\right] \leq q \cdot \epsilon_2$$

holds due to the SEF-OTCMA-security of OTEAE. Similarly, when considering IND-CCA-security, we can prove that

$$\left|\Pr\left[\Gamma^0_{Q,n+1,n} \rightarrow 1\right] - \Pr\left[\Gamma^1_{Q,n+1,n} \rightarrow 1\right]\right| \leq \epsilon_1$$

holds by using the IND-OTCCA security of OTEAE. ■

5. CONSTRUCTION OF OTEAE

In this section, we give two instantiations of OTEAE: a generic one from a secure AEAD and an optimized one based on AES-GCM.

5.1. OTEAE from AEAD

We first give the generic construction of OTEAE from AEAD, which we denote by OTEAE_1. We use a special element $\diamond$ in the associated data domain $\mathsf{AEAD}.\mathcal{A}(\lambda) = \mathsf{OTEAE}_1.\mathcal{A}(\lambda) \cup \{\diamond\}$ with $\diamond$ not in $\mathsf{OTEAE}_1.\mathcal{A}(\lambda)$. We set $\mathsf{OTEAE}_1.\mathcal{K}(\lambda) = \mathsf{AEAD}.\mathcal{K}(\lambda)$ and for convenience we assume that the key length

$\mathcal{B}_2^{\mathsf{ENC},\mathsf{DEC}}(1^\lambda)$:

pick $\mathsf{k}_1,\ldots,\mathsf{k}_{m-1},\mathsf{k} \xleftarrow{\$} \mathsf{OTEAE}.\mathcal{K}(\lambda)$
set $\mathsf{k}_m \leftarrow \perp$
$\mathsf{st}_Q \leftarrow (\mathsf{k}_1,\mathsf{k})$, $\mathsf{st}_{\overline{Q}} \leftarrow (\mathsf{k},\mathsf{k}_1)$
$\mathsf{cnt}_Q^{\mathsf{send}}, \mathsf{cnt}_{\overline{Q}}^{\mathsf{rec}} \leftarrow 0$
$\mathsf{sent}_*^*, \mathsf{received}_*^* \leftarrow \emptyset$, $\mathsf{t}_{\mathsf{test}} \leftarrow \perp$
$z \leftarrow \mathcal{A}^{\mathsf{RATCH},\mathsf{EXP}_{\mathsf{st}},\mathsf{EXP}_{\mathsf{pt}},\mathsf{CHALLENGE}}(1^\lambda)$
if $\neg \mathsf{C}_{\mathsf{clean}}$ **then return** $\perp$
return z

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

$\mathsf{pt}_P \leftarrow \mathsf{pt}$
$\mathsf{ad}_P \leftarrow \mathsf{ad}$
parse $\mathsf{st}_P = (\mathsf{sk},\mathsf{rk})$
if $P = Q$ and $\mathsf{cnt}_Q^{\mathsf{send}} = m-1$ **then**
  $(\mathsf{k}',\mathsf{ct}_P) \leftarrow \mathsf{ENC}(\mathsf{ad}_P,\mathsf{pt}_P)$
  $\mathsf{sk}' \leftarrow \mathsf{k}'$
  $S[P,m,n,\mathsf{pt}_P] \leftarrow \mathsf{k}'$
  $\mathsf{cnt}_Q^{\mathsf{send}} \leftarrow \mathsf{cnt}_Q^{\mathsf{send}} + 1$
else
  $(\mathsf{ct}_P,\mathsf{sk}') \leftarrow \mathsf{OTEAE.Enc}(\mathsf{sk},\mathsf{ad}_P,\mathsf{pt}_P)$
  if $P = Q$ **then**
    $\mathsf{cnt}_Q^{\mathsf{send}} \leftarrow \mathsf{cnt}_Q^{\mathsf{send}} + 1$
    if $\mathsf{cnt}_Q^{\mathsf{send}} \leqslant m-1$ **then**
      $\mathsf{cnt} \leftarrow \mathsf{cnt}_Q^{\mathsf{send}} + 1$
      $\mathsf{sk}' \leftarrow \mathsf{k}_{\mathsf{cnt}}$
$\mathsf{st}_P \leftarrow (\mathsf{sk}',\mathsf{rk})$
append $(\mathsf{ad}_P,\mathsf{pt}_P)$ to $\mathsf{sent}_{\mathsf{pt}}^P$
append $(\mathsf{ad}_P,\mathsf{ct}_P)$ to $\mathsf{sent}_{\mathsf{ct}}^P$
return ct_P

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

$\mathsf{ct}_P \leftarrow \mathsf{ct}$
$\mathsf{ad}_P \leftarrow \mathsf{ad}$
parse $\mathsf{st}_P = (\mathsf{sk},\mathsf{rk})$
if $P = \overline{Q}$ and $\mathsf{cnt}_{\overline{Q}}^{\mathsf{rec}} = m-1$ **then**
  $\mathsf{pt}_P \leftarrow \mathsf{DEC}(\mathsf{ad}_P,\mathsf{pt}_P)$
  if $\mathsf{pt}_P = \perp$ **then**
    return false
  $\mathsf{sk}' \leftarrow S[\overline{P},m,n,\mathsf{pt}_P]$
  $\mathsf{cnt}_{\overline{Q}}^{\mathsf{rec}} \leftarrow \mathsf{cnt}_{\overline{Q}}^{\mathsf{rec}} + 1$
else
  $(\mathsf{pt}_P,\mathsf{rk}') \leftarrow \mathsf{OTEAE.Dec}(\mathsf{rk},\mathsf{ad},\mathsf{ct})$
  if $\mathsf{pt}_P = \perp$ **then return** false
  if $P = \overline{Q}$ **then**
    $\mathsf{cnt}_{\overline{Q}}^{\mathsf{rec}} \leftarrow \mathsf{cnt}_{\overline{Q}}^{\mathsf{rec}} + 1$
    if $\mathsf{cnt}_{\overline{Q}}^{\mathsf{rec}} \leqslant m-1$ **then**
      $\mathsf{cnt} \leftarrow \mathsf{cnt}_{\overline{Q}}^{\mathsf{rec}} + 1$
      $\mathsf{rk}' \leftarrow \mathsf{k}_{\mathsf{cnt}}$
$\mathsf{st}_P \leftarrow (\mathsf{sk},\mathsf{rk}')$
append $(\mathsf{ad}_P,\mathsf{pt}_P)$ to $\mathsf{received}_{\mathsf{pt}}^P$
append $(\mathsf{ad}_P,\mathsf{ct}_P)$ to $\mathsf{received}_{\mathsf{ct}}^P$
return true

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

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

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

return pt_P

Subroutine CHALLENGE(P, ad, pt)

if $\mathsf{t}_{\mathsf{test}} \neq \perp$ **then return** $\perp$
if $b = 0$ **then**
  replace pt by a random string of the same length
$\mathsf{ct} \leftarrow \mathsf{RATCH}(P,$ "send", $\mathsf{ad},\mathsf{pt})$
if $\mathsf{cnt}_Q^{\mathsf{send}} \neq n$ or $P \neq Q$ **then**
  abort
$(\mathsf{t},P,\mathsf{ad},\mathsf{pt},\mathsf{ct})_{\mathsf{test}} \leftarrow (\mathsf{time},P,\mathsf{ad},\mathsf{pt},\mathsf{ct})$
return ct

FIGURE 13. Adversary $\mathcal{B}_2$ against KIND security based on an adversary distinguishing between $\Gamma_{Q,m,n}$ and $\Gamma_{Q,m+1,n}$.

of AEAD is equal to its ciphertext length with input associated data $\diamond$. We define

$$\mathsf{OTEAE}_1.\mathsf{Enc}(\mathsf{k},\mathsf{ad},\mathsf{pt}) = (\mathsf{AEAD.Enc}(\mathsf{k},\diamond,0),\mathsf{AEAD.Enc}(\mathsf{k},\mathsf{ad},\mathsf{pt})),$$

and

$$\mathsf{OTEAE}_1.\mathsf{Dec}(\mathsf{k},\mathsf{ad},\mathsf{ct}) = (\mathsf{AEAD.Enc}(\mathsf{k},\diamond,0),\mathsf{AEAD.Dec}(\mathsf{k},\mathsf{ad},\mathsf{ct}))\text{or}\perp.$$

Security. We prove that, when the underlying AEAD is secure, the OTEAE construction from AEAD is secure.

THEOREM 5.1. *Assume that* AEAD *is* (q,ϵ)-IND-CCA-*secure, then* OTEAE_1 *is* (q,ϵ)-SEF-OTCMA-*secure,* $(q,2\epsilon)$-IND-OTCCA-*secure and* $(q,2\epsilon)$-KIND-*secure.*

Proof. We first consider IND-OTCCA security. We construct two IND-CCA adversaries, $\mathcal{A}_0$ and $\mathcal{A}_1$, for AEAD, which

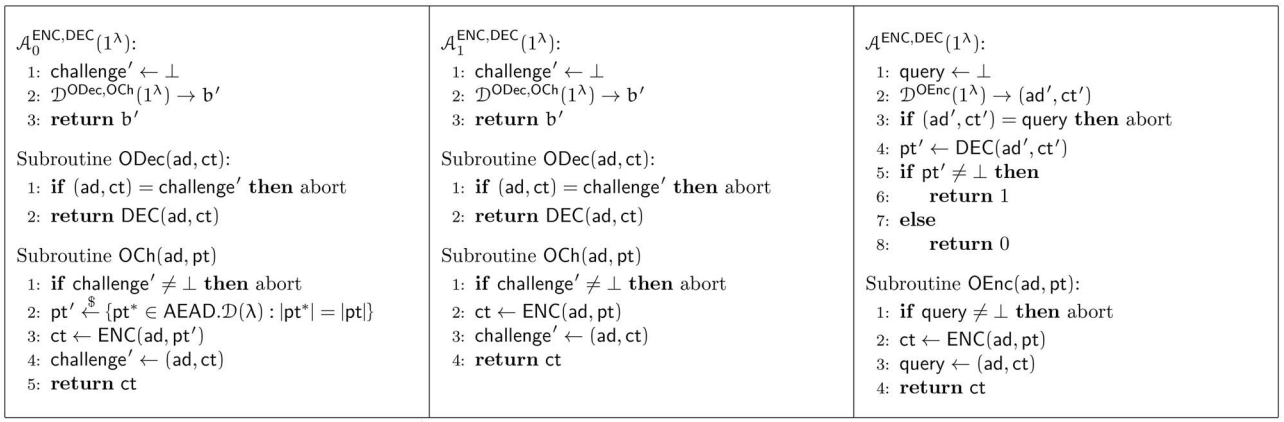


FIGURE 14. IND-CCA adversaries $\mathcal{A}_0$ and $\mathcal{A}_1$ for AEAD based on IND-OTCCA adversary $\mathcal{D}$ (left and middle) and IND-CCA adversary $\mathcal{A}$ based on SEF-OTCMA adversary $\mathcal{D}$ (right) for OTEAE_1.

simulate for IND-OTCCA adversary $\mathcal{D}$, shown in Fig. 14. Adversary $\mathcal{A}_1$ processes $\mathcal{D}$'s queries via its own oracles. $\mathcal{A}_0$ does the same except that when $\mathcal{D}$ makes a challenge query, $\mathcal{A}_0$ first samples a random plaintext, which it uses as input to its oracle ENC. Consider $\mathsf{IND\text{-}CCA}_1$. For each $b \in \{0,1\}$, $\mathcal{A}_b$ perfectly simulates $\mathsf{IND\text{-}OTCCA}_b$ for $\mathcal{D}$. Thus, we have:

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

$$\Pr\left[\mathsf{IND-CCA}_1^{\mathcal{A}_1} \to 1\right] = \Pr\left[\mathsf{IND-OTCCA}_1^{\mathcal{D}} \to 1\right].$$

When $b = 0$, $\mathcal{A}_0$ and $\mathcal{A}_1$ simulate identically distributed games for $\mathcal{D}$, and so:

$$\Pr\left[\mathsf{IND-CCA}_0^{\mathcal{A}_0} \to 1\right] = \Pr\left[\mathsf{IND-CCA}_0^{\mathcal{A}_1} \to 1\right].$$

By application of the triangle inequality, it follows that

$$\begin{aligned}&\left|\Pr\left[\mathsf{IND-OTCCA}_0^{\mathcal{D}} \to 1\right] - \Pr\left[\mathsf{IND-OTCCA}_1^{\mathcal{D}} \to 1\right]\right| \\ &\leq \left|\Pr\left[\mathsf{IND-CCA}_0^{\mathcal{A}_0} \to 1\right] - \Pr\left[\mathsf{IND-CCA}_1^{\mathcal{A}_0} \to 1\right]\right| \\ &+ \left|\Pr\left[\mathsf{IND-CCA}_0^{\mathcal{A}_1} \to 1\right] - \Pr\left[\mathsf{IND-CCA}_1^{\mathcal{A}_1} \to 1\right]\right|,\end{aligned}$$

which is bounded by $2\epsilon(\lambda)$, proving IND-OTCCA security.

Similarly, for any SEF-OTCMA adversary $\mathcal{D}$ for OTEAE, we could construct an IND-CCA adversary $\mathcal{A}$ for AEAD, which is shown in Fig. 14. If $\mathcal{D}$ makes a valid forgery (checked via oracle DEC), then $\mathcal{A}$ outputs 1; otherwise, it outputs 0. When $b = 1$, $\mathcal{A}$ outputs 1 only if $\mathcal{D}$ could make a valid forgery. When $b = 0$, $\mathsf{pt}' = \perp$ always holds and $\mathcal{A}$ always outputs 0. Thus,

$$\Pr\left[\mathsf{IND\text{-}CCA}_1^{\mathcal{A}} \to 1\right] = \Pr\left[\mathsf{SEF\text{-}OTCMA}^{\mathcal{D}} \to 1\right],$$

$$\Pr\left[\mathsf{IND\text{-}CCA}_0^{\mathcal{A}} \to 1\right] = 0.$$

Therefore,

$$\begin{aligned}&\Pr\left[\mathsf{IND\text{-}CCA}_0^{\mathcal{A}} \to 1\right] - \Pr\left[\mathsf{IND\text{-}CCA}_1^{\mathcal{A}} \to 1\right] \\ &= \Pr\left[\mathsf{SEF\text{-}OTCMA}^{\mathcal{D}} \to 1\right],\end{aligned}$$

namely $\mathsf{Adv}(\mathcal{A}) = \mathsf{Adv}(\mathcal{D})$, which is bounded by $\epsilon(\lambda)$.

Finally, we can construct two IND-CCA adversaries $\mathcal{A}$ and $\mathcal{B}$ for AEAD based on the KIND adversary for OTEAE, as is shown in Fig. 15. Then we have

$$\Pr\left[\mathsf{IND\text{-}CCA}_1^{\mathcal{A}} \to 1\right] = \Pr\left[\mathsf{KIND}_1^{\mathcal{D}} \to 1\right],$$

$$\Pr\left[\mathsf{IND\text{-}CCA}_1^{\mathcal{B}} \to 1\right] = \Pr\left[\mathsf{KIND}_0^{\mathcal{D}} \to 1\right].$$

Note also that we have $\Pr\left[\mathsf{IND\text{-}CCA}_0^{\mathcal{A}} \to 1\right] = \Pr\left[\mathsf{IND-CCA}_0^{\mathcal{B}} \to 1\right]$, since $\mathsf{OEnc}/\mathsf{OEnc}'$ output uniformly distributed $(\mathsf{k}',\mathsf{ct})$ values and $\mathsf{ODec}/\mathsf{ODec}'$ always return $\perp$. Thus, we have that:

$$\mathsf{Adv}(\mathcal{D}) \leq \mathsf{Adv}(\mathcal{A}) + \mathsf{Adv}(\mathcal{B}),$$

which is bounded by $2\epsilon(\lambda)$. ■

5.2. OTEAE based on AES-GCM

We also consider a specific simple instantiation of an OTEAE, which is based on AES-128 [25] and the GCM [23]; we call

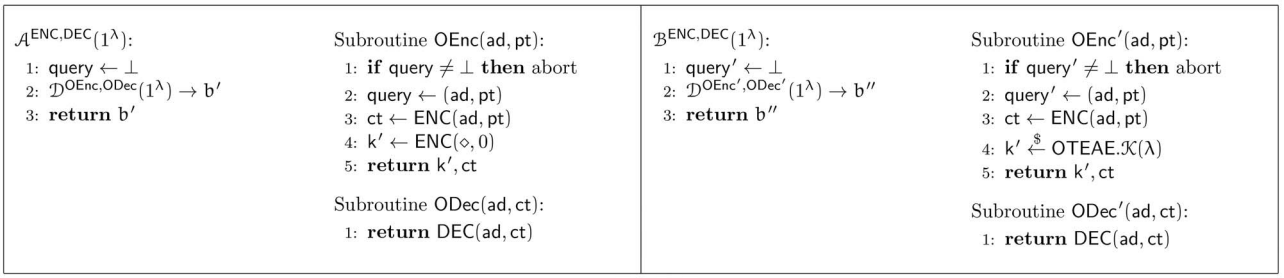


FIGURE 15. IND-CCA adversary $\mathcal{A}$ and $\mathcal{B}$ for AEAD based on KIND adversary $\mathcal{D}$ for OTEAE_1.

this construction OTEAE_2. We note that the construction and analysis for AES-192 and AES-256 is very similar to ours here, and so one could use AES-192 or AES-256 if desired. We first recall relevant details of GCM and AES-GCM. We describe the algorithm for AES-GCM in B.

AEAD-N. We model AES-GCM as a *nonced* AEAD (denote by AEAD-N), which follows the same syntax as AEAD except for introducing an extra input nonce nc:

$$\mathsf{AEAD\text{-}N.Enc}(\mathsf{k},\mathsf{nc},\mathsf{ad},\mathsf{pt}) = \mathsf{ct},$$

$$\mathsf{AEAD\text{-}N.Dec}(\mathsf{k},\mathsf{nc},\mathsf{ad},\mathsf{ct}) = \mathsf{pt} \text{ or } \perp.$$

We say that an adversary is *nonce-respecting* if he never makes two encryption queries with the same nonce.

DEFINITION 5.1. *(*IND-CCA *Security for* AEAD-N*). An* AEAD-N *scheme is* (q,ϵ)-IND-CCA*-secure, if for any PPT nonce-respecting adversary* $\mathcal{A}$ *bounded by* $q(\lambda)$ *queries to oracles, the advantage*

$$\mathit{Adv}(\mathcal{A}) = Pr\left[\mathsf{IND\text{-}CCA}_0^{\mathcal{A}}(1^\lambda) \to 1\right] - Pr\left[\mathsf{IND\text{-}CCA}_1^{\mathcal{A}}(1^\lambda) \to 1\right]$$

of $\mathcal{A}$ *playing the following game is bounded by* $\epsilon(\lambda)$.

Game $\mathsf{IND\text{-}CCA}_b^{\mathcal{A}}(1^\lambda)$:
1: $\mathsf{used} \leftarrow \emptyset$; $\mathsf{challenge} \leftarrow \emptyset$
2: $\mathsf{k} \xleftarrow{\$} \mathsf{AEAD\text{-}N}.\mathcal{K}(\lambda)$
3: $\mathcal{A}^{\mathsf{ENC},\mathsf{DEC}}(1^\lambda) \to b'$
4: **return** b'

Oracle $\mathsf{DEC}(\mathsf{nc},\mathsf{ad},\mathsf{ct})$
1: **if** $(\mathsf{ad},\mathsf{ct}) \in \mathsf{challenge}$ **then return** $\perp$
2: **end if**
3: **if** $b = 0$ **then return** $\perp$
4: **end if**
5: **return** $\mathsf{AEAD\text{-}N.Dec}(\mathsf{k},\mathsf{nc},\mathsf{ad},\mathsf{ct})$

Oracle $\mathsf{ENC}(\mathsf{nc},\mathsf{ad},\mathsf{pt})$
1: **if** $\mathsf{nc} \in \mathsf{used}$ **then** abort
2: **end if**
3: $\mathsf{used} \leftarrow \mathsf{used} \cup \{\mathsf{nc}\}$
4: **if** $\mathsf{b} = 0$ **then**
5: replace pt by a random message of the same length
6: **end if**
7: $\mathsf{ct} \leftarrow \mathsf{AEAD\text{-}N.Enc}(\mathsf{k},\mathsf{nc},\mathsf{ad},\mathsf{pt})$
8: $\mathsf{challenge} \leftarrow \mathsf{challenge} \cup \{(\mathsf{ad},\mathsf{ct})\}$
9: **return** ct

GCM can be considered a mode of operation for a block cipher. AES-GCM indicates GCM instantiated with AES as the underlying symmetric-key block cipher. The GCM key is the block cipher key (the key, for short). The two functions that comprise GCM are called authenticated encryption and authenticated decryption. The authenticated encryption operation AES-GCM.Enc takes as inputs a secret key k, initialization vector (nonce) nc, associated data ad, and a plaintext pt, and outputs a ciphertext ct_0 and an authentication tag tag, namely $\mathsf{AES\text{-}GCM.Enc}(\mathsf{k},\mathsf{nc},\mathsf{ad},\mathsf{pt}) = (\mathsf{ct}_0,\mathsf{tag})$. Note that in the following, we merge the tag into the ciphertext, and so we write:

$$\mathsf{AES\text{-}GCM.Enc}(\mathsf{k},\mathsf{nc},\mathsf{ad},\mathsf{pt}) = \mathsf{ct},$$

where $\mathsf{ct} = (\mathsf{ct}_0,\mathsf{tag})$. The authenticated decryption operation AES-GCM.Dec takes k, nc, ad and ct and outputs either the plaintext value pt or the special symbol $\perp$ that indicates that its inputs are not authentic, namely

$$\mathsf{AES\text{-}GCM.Dec}(\mathsf{k},\mathsf{nc},\mathsf{ad},\mathsf{ct}) = \mathsf{pt or} \perp.$$

For fixed values of k, nc, ad and pt, the calls $\mathsf{AES\text{-}GCM.Enc}(\mathsf{k},\mathsf{nc},\mathsf{ad},\mathsf{pt}) \to \mathsf{ct}$ and $\mathsf{AES\text{-}GCM.Dec}(\mathsf{k},\mathsf{nc},\mathsf{ad},\mathsf{ct})$ make the same underlying calls to AES.Enc, and make no calls to AES.Dec. We assume that the nonce nc is 96 bits long: AES-GCM operations are more efficient in this case and this is recommended in practice by bodies such as NIST [26]. For

such a nc and a ciphertext of length ℓ, the following calls to $\mathsf{AES.Enc}$ are made:

- One call $\mathsf{AES.Enc}(\mathsf{k}, 0^{128})$; and
- $m = \lfloor \frac{\ell}{128} \rfloor + 2$ calls of the form $\mathsf{AES.Enc}(\mathsf{k}, \mathsf{nc} \mid\mid ((0^{31} \mid\mid 1 + i) \mod 2^{32}))$ for $i = 0, ..., m - 1$.

Observation. Observe that, assuming $|\mathsf{nc}| = 96$, $\mathsf{AES.Enc}(\mathsf{k}, 1^{128})$ is only called in $\mathsf{AES\text{-}GCM.Enc}$ or $\mathsf{AES\text{-}GCM.Dec}$ when $\mathsf{nc} = 1^{96}$.

We now define OTEAE_2 as follows: For $\mathsf{OTEAE}_2.\mathsf{Enc}$, it takes a key k in $\{0,1\}^{128}$, associated data ad and a message pt, and returns the ciphertext ct output by AES-GCM encryption with $\mathsf{nc} = 0^{96}$, together with an updated new key $\mathsf{k}' = \mathsf{AES.Enc}(\mathsf{k}, 1^{128})$, which is the ciphertext of plaintext 1^{128} encrypted by AES. Similarly, for $\mathsf{OTEAE}_2.\mathsf{Dec}$, it recovers the message pt by AES-GCM decryption and updates the key state to $\mathsf{k}' = \mathsf{AES.Enc}(\mathsf{k}, 1^{128})$. Formally, we have:

$$\mathsf{OTEAE}_2.\mathsf{Enc}(\mathsf{k}, \mathsf{ad}, \mathsf{pt}) =$$

$$(\mathsf{AES.Enc}(\mathsf{k}, 1^{128}), \mathsf{AES\text{-}GCM.Enc}(\mathsf{k}, 0^{96}, \mathsf{ad}, \mathsf{pt})),$$

and

$$\mathsf{OTEAE}_2.\mathsf{Dec}(\mathsf{k}, \mathsf{ad}, \mathsf{ct}) = \begin{cases} \perp, & (x = \perp) \\ (\mathsf{AES.Enc}(\mathsf{k}, 1^{128}), x), & (x \neq \perp) \end{cases}$$

where $x = \mathsf{AES\text{-}GCM.Dec}(\mathsf{k}, 0^{96}, \mathsf{ad}, \mathsf{ct})$.

Security. We introduce the PRP assumption, modelled as a game in which the adversary $\mathcal{A}$ is given access to a permutation oracle and is challenged to determine whether it is playing with the block cipher BC with a randomly selected key, or a random permutation RP.

DEFINITION 5.2. *(PRP). A block cipher* BC *is* (q, ϵ)-PRP-*secure, if for any PPT adversary* $\mathcal{A}$ *bounded by* $q(\lambda)$ *queries to oracles, the advantage*

$$Pr\left[\mathsf{PRP}^{\mathsf{BC}}_{\mathcal{A}}(1^\lambda) \to 1\right] - Pr\left[\mathsf{PRP}^{\mathsf{RP}}_{\mathcal{A}}(1^\lambda) \to 1\right]$$

of $\mathcal{A}$ *playing the following game is bounded by* $\epsilon(\lambda)$.

Game $\mathsf{PRP}^{\mathcal{O}}_{\mathcal{A}}(1^\lambda)$:
1: $\mathsf{k} \leftarrow \mathsf{BC}.\mathcal{K}(\lambda)$
2: $\mathsf{RP} \leftarrow \mathsf{Perm}(\mathsf{BC}.\mathcal{D}(\lambda))$
3: $\mathcal{A}^{\mathcal{O}}(1^\lambda) \to b'$
4: **return** b'

Oracle $\mathcal{O}(\mathsf{pt})$
1: $\mathsf{BC.Enc}(\mathsf{k}, \mathsf{pt}) \to \mathsf{ct}$
2: **if** $\mathcal{O} = \mathsf{RP}$ **then**
3: $\quad \mathsf{ct} \leftarrow \mathsf{RP}(\mathsf{pt})$
4: **end if**
5: **return** ct

*$\mathsf{Perm}(\mathcal{D})$ is the set of all permutations over domain $\mathcal{D}$.

In [27], McGrew and Viega aim to prove, assuming the underlying block cipher (AES in our case) is a secure PRP, that GCM instantiated with this block cipher is an $\mathsf{IND-CCA}$-secure $\mathsf{AEAD-N}$. Iwata *et al.* [28] note flaws in their proofs and provide new reductions to PRP security which are tighter than those of McGrew and Viega assuming the use of 96-bit nonces.

We first state a result that adapts the results of Iwata et al. to our definition of $\mathsf{AEAD-NIND\text{-}CCA}$ security:

LEMMA 5.1. *([28]). Suppose that* AES *is a* (q, ϵ_1)-*secure* PRP. *Given that only 96-bit nonces are used and 128-bit tags are output,* $\mathsf{AES-GCM}$ *is a* $(q, 3\epsilon_1 + \epsilon_2)$-$\mathsf{IND\text{-}CCA}$-*secure* $\mathsf{AEAD\text{-}N}$, *where* $\epsilon_2 = \frac{3(\sigma+q+1)^2}{2^{129}} + \frac{q\cdot(\ell_A+1)}{2^{128}}$ *and* σ *denotes the total number of encrypted blocks with a total bit length of at most* ℓ_A.

We proceed to proving security:

THEOREM 5.2. *Assume that* $\mathsf{AES.Enc}$ *is a* (q, ϵ_1)-*secure* PRP, *(then* $\mathsf{AES\text{-}GCM}$ *is a* (q, ϵ)-$\mathsf{IND\text{-}CCA}$-*secure* $\mathsf{AEAD\text{-}N}$, *where* $\epsilon = 3\epsilon_1 + \epsilon_2$ *and* ϵ_2 *is defined above), then* OTEAE_2 *is* (q, ϵ)-$\mathsf{SEF\text{-}OTCMA}$-*secure,* $(q, 2\epsilon)$-$\mathsf{IND\text{-}OTCCA}$-*secure and* $(q, 2\epsilon + \epsilon')$-$\mathsf{KIND}$-*secure where* $\epsilon' = \frac{3(\sigma+q+2)}{2^{127}}$.

Proof. We first consider $\mathsf{IND\text{-}OTCCA}$ and $\mathsf{SEF\text{-}OTCMA}$ security. Note that the key k' output by $\mathsf{OTEAE}_2.\mathsf{Enc}$ and $\mathsf{OTEAE}_2.\mathsf{Dec}$ is not provided to an adversary playing either game. Thus, parallel to the proof of Theorem 5.1, for any $\mathsf{IND\text{-}OTCCA}$ adversary $\mathcal{D}'$ for OTEAE_2, we could construct $\mathsf{IND\text{-}CCA}$ adversaries $\mathcal{A}_0$ and $\mathcal{A}_1$ for $\mathsf{AEAD\text{-}N}$ (refer to Fig. 16). Similarly, for any $\mathsf{SEF-OTCMA}$ adversary $\mathcal{D}'$ for OTEAE_2, we could construct $\mathsf{IND\text{-}CCA}$ adversaries $\mathcal{A}$ for $\mathsf{AEAD\text{-}N}$ (refer to Fig. 16). By the same arguments as in the proof of Theorem 5.1, the claimed security follows.

We now consider KIND security, which is more involved to prove than the previous two security notions. Let $\mathsf{KIND}^{\mathcal{A}}_b$ be the KIND game played by adversary $\mathcal{A}$ with respect to OTEAE_2 and bit b. Define the game Γ as in KIND_0 except that the output of the first call to ENC comprises $(\mathsf{k}', \mathsf{ct}')$, where k' is uniformly sampled and ct' is the encryption of a uniformly sampled plaintext pt' under $\mathsf{AES\text{-}GCM}$, and the output of DEC is always $\perp$. We present Γ in Fig. 17.

We proceed by upper bounding values p_0 and p_1, where:

$$p_0 = \left|\Pr\left[\mathsf{KIND}^{\mathcal{A}}_0(1^\lambda) \to 1\right] - \Pr\left[\Gamma^{\mathcal{A}}(1^\lambda) \to 1\right]\right|,$$

and

$$p_1 = \left|\Pr\left[\mathsf{KIND}^{\mathcal{A}}_1(1^\lambda) \to 1\right] - \Pr\left[\Gamma^{\mathcal{A}}(1^\lambda) \to 1\right]\right|.$$

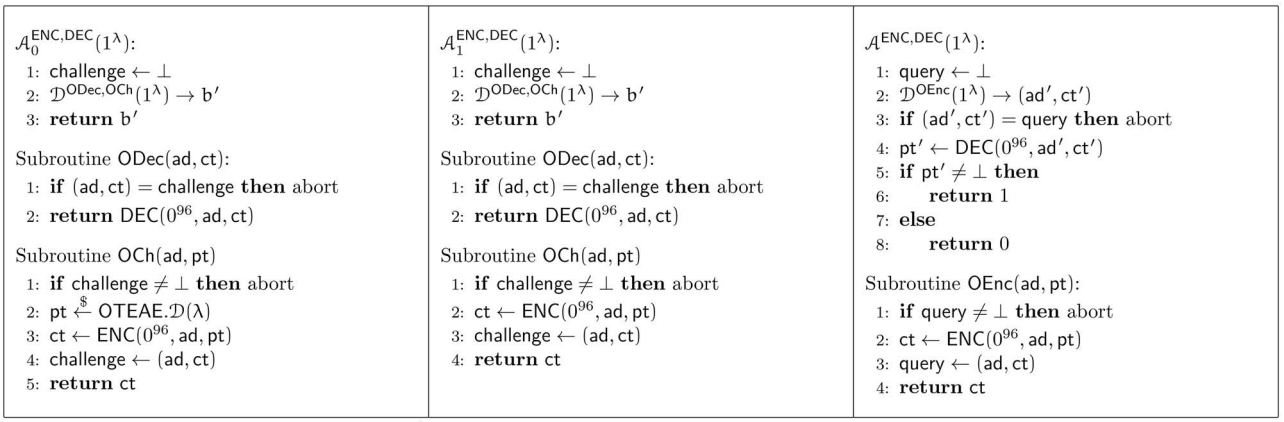


FIGURE 16. IND-CCA adversaries $\mathcal{A}_0$ and $\mathcal{A}_1$ for AEAD-N based on IND-OTCCA adversary $\mathcal{D}$ (left and middle) and IND-CCA adversary $\mathcal{A}$ based on SEF-OTCMA adversary $\mathcal{D}$ (right) for OTEAE_2.

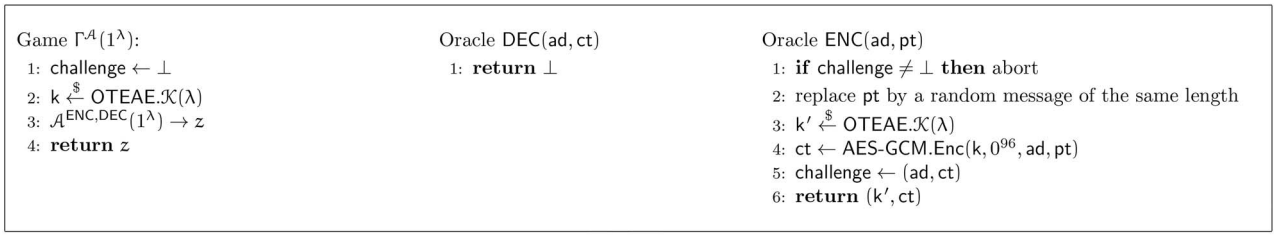


FIGURE 17. OTEAE_2KIND-security: hybrid Γ.

We show that $p_0 \leq \epsilon$ and $p_1 \leq \epsilon + \epsilon'$. Then, by applying the triangle inequality, we obtain:

$$\left|\Pr\left[\mathsf{KIND}_0^{\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathsf{KIND}_1^{\mathcal{A}}(1^\lambda) \to 1\right]\right| \leq 2\epsilon + \epsilon',$$

completing the proof.

We first bound p_0. Let $\mathcal{B}_0$ be an IND-CCA adversary with respect to AEAD-NAES-GCM who simulates for adversary $\mathcal{A}$. $\mathcal{B}_0$ is defined in Fig. 18. In particular, $\mathcal{B}_0$ samples k′ uniformly (as in Γ), and sets ct by calling $\mathsf{ENC}(0^{96}, \mathsf{ad}, \mathsf{pt})$. It can thus be seen that $\mathcal{B}_0$ perfectly simulates KIND_0 when $b = 1$ and Γ when $b = 0$ for $\mathcal{A}$. We thus have:

$$\Pr\left[\mathsf{IND\text{-}CCA}_0^{\mathcal{B}_0}(1^\lambda) \to 1\right] = \Pr\left[\mathsf{KIND}_0^{\mathcal{A}}(1^\lambda) \to 1\right],$$

and

$$\Pr\left[\mathsf{IND\text{-}CCA}_1^{\mathcal{B}_0}(1^\lambda) \to 1\right] = \Pr\left[\Gamma^{\mathcal{A}}(1^\lambda) \to 1\right].$$

By the IND-CCA security of AES-GCM as an AEAD-N, it follows that $p_0 \leq \epsilon$.

We then bound p_1. Let $\mathcal{B}_1$ be a PRP distinguisher, which is defined in Fig. 18. $\mathcal{B}_1$ simulates for adversary $\mathcal{A}$ as in KIND_1 except that AES-GCM and AES are simulated by calls to the PRP oracle $\mathcal{O}$. In particular, simulating AES-GCM operations requires multiple calls to $\mathcal{O}$. This is possible since AES−GCM only uses the input key k in AES.Enc calls (c.f. B). We claim that $p_1 \leq \epsilon + \epsilon$'.

Towards proving this, we consider the proofs of AES-GCM security from Iwata *et al.* [28]. In particular, the authors reduce security to the presumed PRP security of AES. The authors separately prove that GCM is secure with respect to 1) an IND-CPA game and 2) an authenticity game AUTH where the adversary may query both the encryption and decryption oracles, who wins if the decryption oracle outputs pt $\neq \perp$ with respect to input (ct, ad) where ct was not output by a call to the encryption oracle with associated data ad. They do so by a reduction to the PRP game.

Each proof bounds the adversary's advantage by the probability of a collision on inputs to AES.Enc (which is zero in our case). The proof for AUTH security additionally bounds the probability of forging a MAC of the form $\mathsf{AES.Enc}(\mathsf{k}, \mathsf{nc}||0^{31}1) \oplus \mathsf{GHASH}(L, \mathsf{ad}, \mathsf{ct}_0)$ for fixed $L = \mathsf{AES.Enc}(\mathsf{k}, 0^{128})$ where GHASH is a polynomial MAC[3] . Both proofs then invoke the standard PRP/PRF switching

[3] Essentially, a polynomial over GF(2^{128}) with coefficients derived from ad and ct is evaluated at input value L.

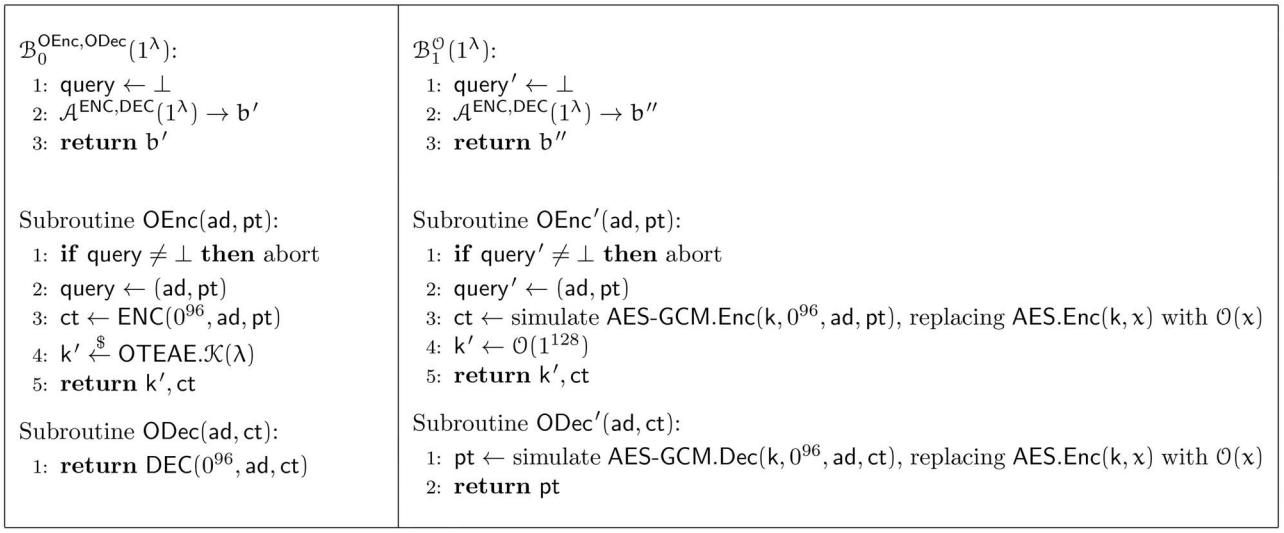


FIGURE 18. IND-CCA adversary $\mathcal{B}_0$ based on Γ/KIND$_0$ adversary $\mathcal{A}$ (left) and PRP adversary $\mathcal{B}_1$ based on Γ/KIND$_1$ adversary $\mathcal{A}$ (right).

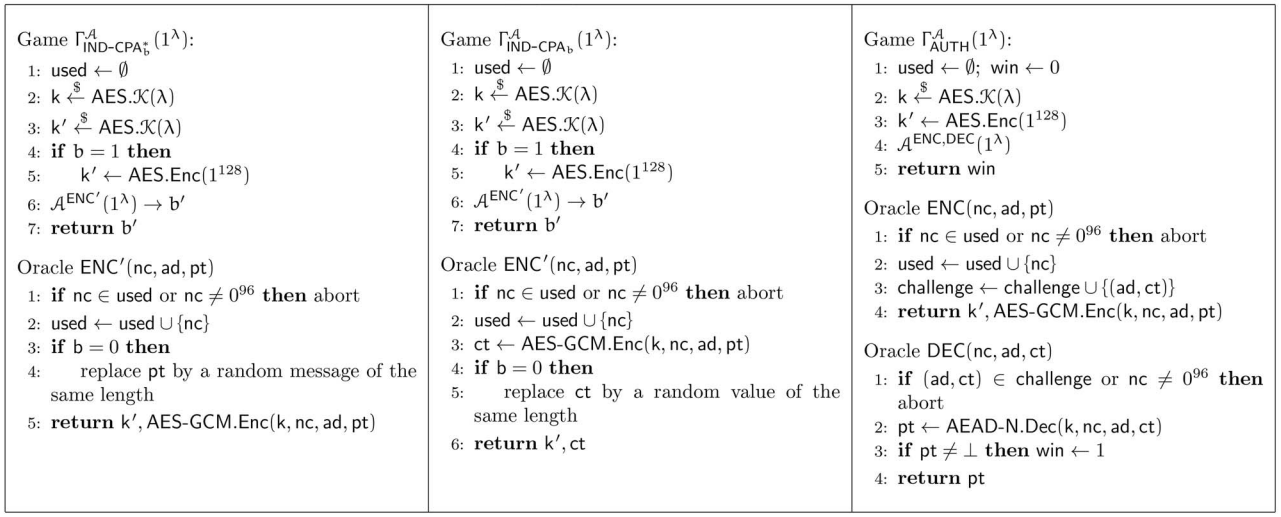


FIGURE 19. OTEAE$_2$KIND-security: $\Gamma_{\mathsf{IND-CPA}^*}$, $\Gamma_{\mathsf{IND-CPA}}$ and Γ_{AUTH} games.

lemma [29]. The insight to prove security for OTEAE$_2$ is that the aforementioned probabilities are not affected by the additional call to AES.Enc(k, 1^{128}), since AES.Enc(k, 1^{128}) is never called by AES-GCM in our setting where nc $= 0^{96}$ always (c.f. B). In particular, the call AES.Enc(k, 1^{128}) only affects the number of queries made to the PRP oracle which is only relevant for the application of the switching lemma (increasing q and σ by 1). Thus, we can mirror the reductions of Iwata et al. to PRP security to simulate except embedding an additional call to the PRP oracle with argument 1^{128} and arrive at the result.

We introduce games $\Gamma_{\mathsf{IND\text{-}CPA}}$ and Γ_{AUTH}, shown in Fig. 19. These games are identical to those of Iwata et al., except that an additional value k$'$ is computed and output by the encryption oracle, and also that nonces nc $\neq 0^{96}$ are rejected as input to each oracle. We define PRP adversaries $\mathcal{C}$ and $\mathcal{D}$ who simulate $\Gamma_{\mathsf{IND\text{-}CPA}}$ and Γ_{AUTH}, respectively, for adversary $\mathcal{A}$, shown in Fig. 20. The reductions are identical to those of Iwata et al. except for the additional oracle call $\mathcal{O}(1^{128})$ made. Recall that σ is the total number of encrypted blocks by AES-GCM calls and q is the number of oracle queries made by the adversary. The authors argue that by the construction of GCM the probability to distinguish between GCM instantiated with a uniform permutation or a uniform function is upper bounded by $\frac{(\sigma+q+1)^2}{2^{129}}$. Applying their results on the probability of a collision on AES in AES-GCM and forging as described

$\mathcal{C}^{\mathcal{O}}(1^\lambda)$:
1: $\mathsf{query} \leftarrow \perp$
2: $\mathcal{A}^{\mathsf{ENC}'}(1^\lambda) \to \mathsf{b}'$
3: **return** b'

Subroutine $\mathsf{OEnc}(\mathsf{ad}, \mathsf{pt})$:
1: **if** $\mathsf{query}' \neq \perp$ **then** abort
2: $\mathsf{query}' \leftarrow (\mathsf{ad}, \mathsf{pt})$
3: $\mathsf{ct} \leftarrow$ simulate $\mathsf{AES\text{-}GCM.Enc}(\mathsf{k}, 0^{96}, \mathsf{ad}, \mathsf{pt})$, replacing $\mathsf{AES.Enc}(\mathsf{k}, \mathsf{x})$ with $\mathcal{O}(\mathsf{x})$
4: $\mathsf{k}' \leftarrow \mathcal{O}(1^{128})$
5: **return** k', ct

$\mathcal{D}^{\mathcal{O}}(1^\lambda)$:
1: $\mathsf{query}' \leftarrow \perp$; $\mathsf{win} \leftarrow 0$
2: $\mathcal{A}^{\mathsf{ENC},\mathsf{DEC}}(1^\lambda)$
3: **return** win

Subroutine $\mathsf{ENC}(\mathsf{ad}, \mathsf{pt})$:
1: **if** $\mathsf{query}' \neq \perp$ **then** abort
2: $\mathsf{query}' \leftarrow (\mathsf{ad}, \mathsf{pt})$
3: $\mathsf{ct} \leftarrow$ simulate $\mathsf{AES\text{-}GCM.Enc}(\mathsf{k}, 0^{96}, \mathsf{ad}, \mathsf{pt})$, replacing $\mathsf{AES.Enc}(\mathsf{k}, \mathsf{x})$ with $\mathcal{O}(\mathsf{x})$
4: $\mathsf{k}' \leftarrow \mathcal{O}(1^{128})$
5: **return** k', ct

Subroutine $\mathsf{DEC}(\mathsf{ad}, \mathsf{ct})$:
1: $\mathsf{pt} \leftarrow$ simulate $\mathsf{AES\text{-}GCM.Dec}(\mathsf{k}, 0^{96}, \mathsf{ad}, \mathsf{ct})$, replacing $\mathsf{AES.Enc}(\mathsf{k}, \mathsf{x})$ with $\mathcal{O}(\mathsf{x})$
2: **if** $\mathsf{pt} \neq \perp$ **then** $\mathsf{win} \leftarrow 1$
3: **return** pt

FIGURE 20. $\Gamma_{\mathsf{IND\text{-}CPA}}$ and Γ_{AUTH} adversaries $\mathcal{C}$ and $\mathcal{D}$ based on PRP adversary $\mathcal{A}$.

$\mathcal{A}'^{\mathsf{ENC}}(1^\lambda)$:
1: $\mathsf{query} \leftarrow \perp$
2: $\mathcal{A}^{\mathsf{ENC}'}(1^\lambda) \to \mathsf{b}'$
3: **return** b'

Subroutine $\mathsf{ENC}'(\mathsf{ad}, \mathsf{pt})$:
1: **if** $\mathsf{query}' \neq \perp$ **then** abort
2: $\mathsf{query}' \leftarrow (\mathsf{ad}, \mathsf{pt})$
3: $(\mathsf{k}', \mathsf{ct}) \leftarrow \mathsf{ENC}(\mathsf{ad}, \mathsf{pt})$
4: **return** k', ct

$\mathcal{A}''^{\mathsf{ENC}}(1^\lambda)$:
1: $\mathsf{query} \leftarrow \perp$
2: $\mathcal{A}^{\mathsf{ENC}'}(1^\lambda) \to \mathsf{b}'$
3: **return** b'

Subroutine $\mathsf{ENC}'(\mathsf{ad}, \mathsf{pt})$:
1: **if** $\mathsf{query}' \neq \perp$ **then** abort
2: $\mathsf{query}' \leftarrow (\mathsf{ad}, \mathsf{pt})$
3: replace pt by a random value of the same length
4: $(\mathsf{k}', \mathsf{ct}) \leftarrow \mathsf{ENC}(\mathsf{ad}, \mathsf{pt})$
5: **return** k', ct

FIGURE 21. $\Gamma_{\mathsf{IND\text{-}CPA}}$ adversaries $\mathcal{A}'$ and $\mathcal{A}''$ based on $\Gamma_{\mathsf{IND\text{-}CPA}^*}$ adversary $\mathcal{A}$.

above, and accounting for the additional AES encryption (on a single block) in this bound, we have:

$$\Pr\left[\Gamma^{\mathcal{A}}_{\mathsf{IND\text{-}CPA}_0}(1^\lambda) \to 1\right] - \Pr\left[\Gamma^{\mathcal{A}}_{\mathsf{IND\text{-}CPA}_1}(1^\lambda) \to 1\right] \leq \epsilon_1 + \frac{(\sigma + q + 3)^2}{2^{129}},$$

and

$$\Pr\left[\Gamma^{\mathcal{A}}_{\mathsf{AUTH}}(1^\lambda) \to 1\right] \leq \epsilon_1 + \frac{q \cdot (\ell_A + 1)}{2^{128}} + \frac{(\sigma + q + 3)^2}{2^{129}}.$$

We also define a game $\Gamma_{\mathsf{IND\text{-}CPA}^*}$, shown in Fig. 19. When $b = 1$, $\Gamma_{\mathsf{IND\text{-}CPA}^*}$ and $\Gamma_{\mathsf{IND\text{-}CPA}}$ are identically distributed. Note that when $b = 0$, $\Gamma_{\mathsf{IND\text{-}CPA}^*}$ samples pt uniformly which it then encrypts, whereas $\Gamma_{\mathsf{IND\text{-}CPA}}$ samples the *ciphertext* (including the tag) uniformly. Note that the output of the encryption oracles in these games are, in general, distributed differently.

Let $\mathcal{A}'$ and $\mathcal{A}''$ be $\Gamma_{\mathsf{IND\text{-}CPA}}$ adversaries who simulate for $\Gamma_{\mathsf{IND\text{-}CPA}^*}$ adversary $\mathcal{A}$. $\mathcal{A}'$ and $\mathcal{A}''$ are defined in Fig. 21. Firstly, note that when $b = 0$, $\mathcal{A}'$ and $\mathcal{A}''$ simulate the same game for $\mathcal{A}$, since the input pt is ignored by their respective challenger. Note that we also have:

$$\Pr\left[\Gamma^{\mathcal{A}'}_{\mathsf{IND\text{-}CPA}_1}(1^\lambda) \to 1\right] = \Pr\left[\Gamma^{\mathcal{A}}_{\mathsf{IND\text{-}CPA}^*_1}(1^\lambda) \to 1\right],$$

$$\Pr\left[\Gamma^{\mathcal{A}''}_{\mathsf{IND\text{-}CPA}_1}(1^\lambda) \to 1\right] = \Pr\left[\Gamma^{\mathcal{A}}_{\mathsf{IND\text{-}CPA}^*_0}(1^\lambda) \to 1\right].$$

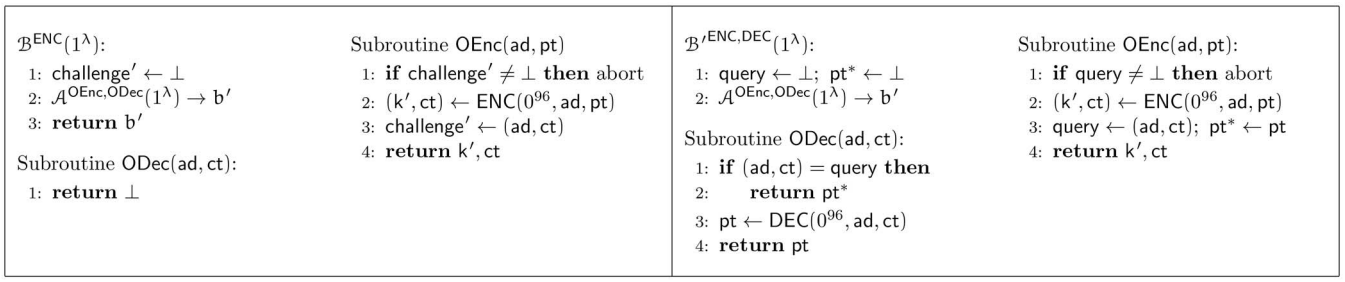


FIGURE 22. $\Gamma_{\mathsf{IND\text{-}CPA}^*}$ adversary $\mathcal{B}$ for AEAD based on KIND_1/Γ adversary $\mathcal{A}$ (left) and Γ_{AUTH} adversary $\mathcal{B}'$ based on KIND_1/Γ adversary $\mathcal{A}$ (right) for OTEAE_2.

Thus, we have:

$$\begin{aligned}
&\left|\Pr\left[\Gamma^{\mathcal{A}}_{\mathsf{IND\text{-}CPA}^*_0}(1^\lambda) \to 1\right] - \Pr\left[\Gamma^{\mathcal{A}}_{\mathsf{IND\text{-}CPA}^*_1}(1^\lambda) \to 1\right]\right| \\
&= \left|\Pr\left[\Gamma^{\mathcal{A}''}_{\mathsf{IND\text{-}CPA}_1}(1^\lambda) \to 1\right] - \Pr\left[\Gamma^{\mathcal{A}'}_{\mathsf{IND\text{-}CPA}_1}(1^\lambda) \to 1\right]\right| \\
&\le \left|\Pr\left[\Gamma^{\mathcal{A}'}_{\mathsf{IND\text{-}CPA}_1}(1^\lambda) \to 1\right] - \Pr\left[\Gamma^{\mathcal{A}'}_{\mathsf{IND\text{-}CPA}_0}(1^\lambda) \to 1\right]\right| \\
&+ \left|\Pr\left[\Gamma^{\mathcal{A}''}_{\mathsf{IND\text{-}CPA}_1}(1^\lambda) \to 1\right] - \Pr\left[\Gamma^{\mathcal{A}''}_{\mathsf{IND\text{-}CPA}_0}(1^\lambda) \to 1\right]\right| \\
&\le 2 \cdot \left(\epsilon_1 + \frac{(\sigma+q+3)^2}{2^{129}}\right)
\end{aligned}$$

Before concluding the proof, we first show that there exist adversaries $\mathcal{B}$ and $\mathcal{B}'$ such that:

$$\begin{aligned}
&\left|\Pr\left[\mathsf{KIND}^{\mathcal{A}}_1(1^\lambda) \to 1\right] - \Pr\left[\Gamma^{\mathcal{A}}(1^\lambda) \to 1\right]\right| \\
&\le \left|\left[\Gamma^{\mathcal{B}}_{\mathsf{IND\text{-}CPA}^*_0}(1^\lambda) \to 1\right] - \Pr\left[\Gamma^{\mathcal{B}}_{\mathsf{IND\text{-}CPA}^*_1}(1^\lambda) \to 1\right]\right| \\
&+ \Pr\left[\Gamma^{\mathcal{B}'}_{\mathsf{AUTH}}(1^\lambda) \to 1\right]
\end{aligned}$$

Let $\mathcal{B}$ be a $\Gamma_{\mathsf{IND\text{-}CPA}^*}$ adversary and $\mathcal{B}'$ be a Γ_{AUTH} adversary. These adversaries simulate for KIND_1/Γ adversary $\mathcal{A}$ and are defined in Fig. 22.

Firstly, note that Γ oracle DEC always returns $\perp$. Then, $\mathcal{B}$ simulates for $\mathcal{A}$ playing Γ perfectly given $b = 0$, since in particular $\mathcal{B}$ can simulate decryption queries. Similarly, $\mathcal{B}'$ simulates for $\mathcal{A}$ playing KIND_1 perfectly.

Thus, we have:

$$\begin{aligned}
&\Pr\left[\mathsf{KIND}^{\mathcal{A}}_1(1^\lambda) \to 1\right] - \Pr\left[\Gamma^{\mathcal{A}}(1^\lambda) \to 1\right] \\
&= \Pr\left[\Gamma^{\mathcal{B}'}_{\mathsf{AUTH}}(1^\lambda) \to 1\right] - \Pr\left[\Gamma^{\mathcal{B}}_{\mathsf{IND\text{-}CPA}^*_0}(1^\lambda) \to 1\right] \\
&\le \Pr\left[\Gamma^{\mathcal{B}'}_{\mathsf{AUTH}}(1^\lambda) \to 1\right] \\
&+ \Pr\left[\Gamma^{\mathcal{B}}_{\mathsf{IND\text{-}CPA}^*_0}(1^\lambda) \to 1\right] - \Pr\left[\Gamma^{\mathcal{B}}_{\mathsf{IND\text{-}CPA}^*_0}(1^\lambda) \to 1\right].
\end{aligned}$$

The result follows. Using this result, we obtain:

$$\begin{aligned}
p_1 &= \left|\Pr\left[\mathsf{KIND}^{\mathcal{A}}_1(1^\lambda) \to 1\right] - \Pr\left[\Gamma^{\mathcal{A}}(1^\lambda) \to 1\right]\right| \\
&\le 3\epsilon_1 + \frac{3(\sigma+q+3)^2}{2^{129}} + \frac{q \cdot (\ell_A+1)}{2^{128}} \\
&= \epsilon + \frac{3(\sigma+q+3)^2}{2^{129}} - \frac{3(\sigma+q+1)^2}{2^{129}} \\
&= \epsilon + \frac{3 \cdot 2 \cdot (2\sigma+2q+4)}{2^{129}} \\
&= \epsilon + \frac{3(\sigma+q+2)}{2^{127}} \\
&= \epsilon + \epsilon'.
\end{aligned}$$

■

6. IMPLEMENTATION AND COMPARISON WITH LITEARCAD

In this section, we discuss a preliminary implementation of EtH and EtUK, and compare their software performance with that of several existing ratcheting protocols, including

- Related lightweight constructions ARCAD-DV and liteARCAD [16].
- HIBE-based protocols PR [14] and JS [15].
- The public-key cryptography-based JMM proposal [17].

- The Signal reinterpretation and an extended protocol with signatures and public-key encryption ACD, ACD-PK [8].

All schemes are implemented in Go and measured with its built-in benchmarking suite[4] on a fifth-generation 1.4 GHz Intel Core i5 dual-core laptop processor that is comparable in performance to a contemporary smartphone. The source code of all investigated schemes is publicly available.[5] The EtH

4 https://golang.org/pkg/testing/.

5 https://github.com/qantik/ratcheted. Instructions to run the experiments can be found at https://github.com/dcol97/tcj_sarcad_data.

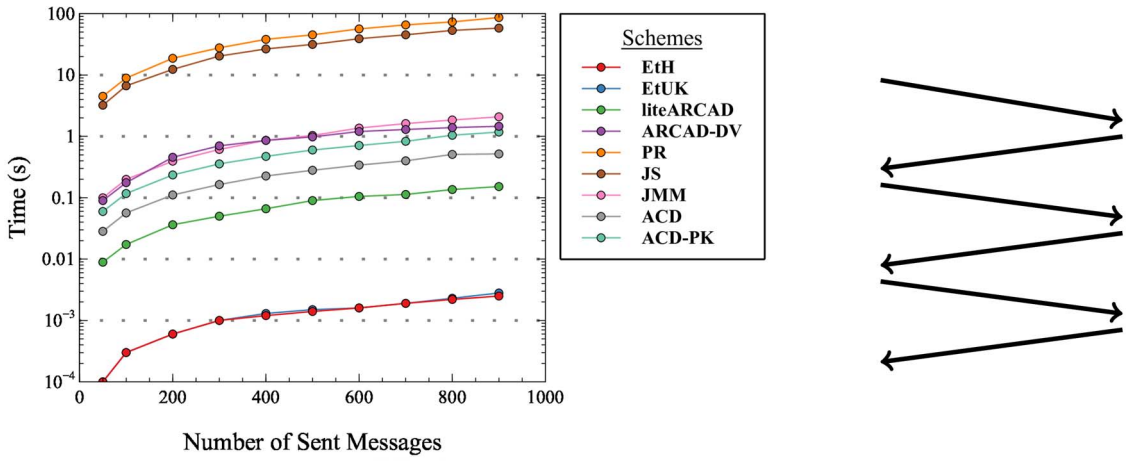


FIGURE 23. Alternating traffic.

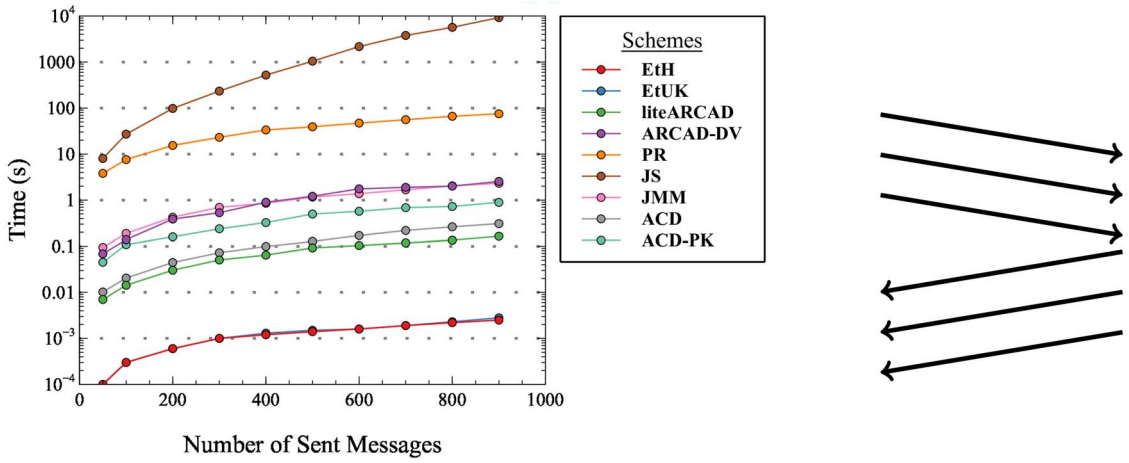


FIGURE 24. Unidirectional traffic.

protocol is instantiated with AES-GCM, provided by the Go standard library, as the OTAE scheme and SHA-256 (truncated to 128 bits) as the hash function H. EtUK (recall Section 4.1)is bootstrapped using OTEAE_2 that combines AES-GCM with a key update through an additional AES-128.Enc invocation.

In line with [16], we consider three canonical message patterns, as depicted in Figures 23, 24 and 25 alongside our runtime benchmarks.

- **Alternating.** Alice and Bob take turns sending messages. Alice sends the odd-numbered messages and Bob sends the even-numbered messages.
- **Unidirectional.** Alice first sends $\frac{n}{2}$ messages to Bob, and after receiving them Bob responds with the remaining $\frac{n}{2}$ messages.
- **Deferred unidirectional.** Alice first sends $\frac{n}{2}$ messages to Bob but before he receives them, Bob sends $\frac{n}{2}$ messages to Alice.

We note that for all three communication patters, EtH (as well as EtUK) exhibits similar performance. The SARCAD schemes have constant sending/receiving time complexity in each case. In the case of EtH, sending/receiving one message requires a single AES-GCM invocation together with one SHA-256 evaluation, whereas for EtUK, sending/receiving one message necessitates one AES-GCM call and one AES-128 encryption.

Comparing EtH with EtUK, the sole difference is that a SHA-256 call is replaced by an encryption with AES-128. From the runtime benchmarks, we do not find much difference

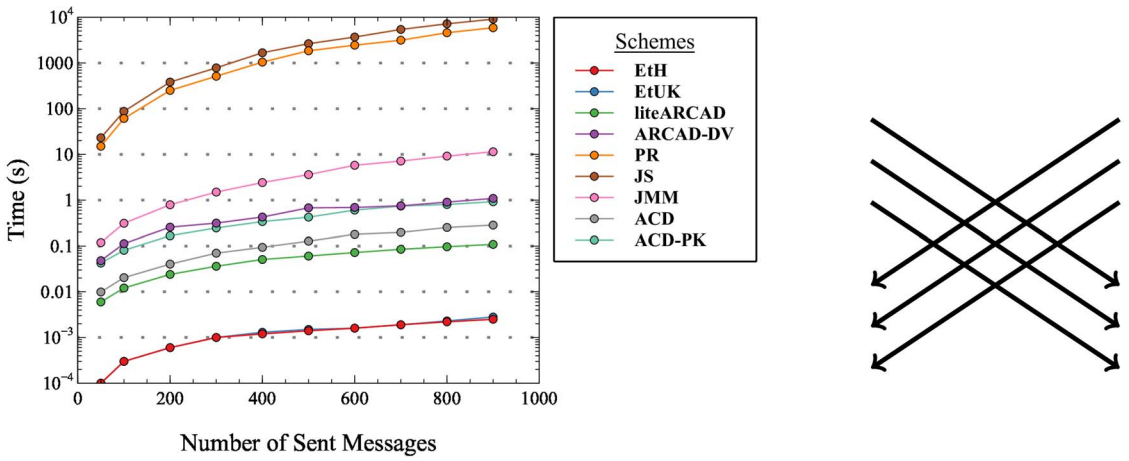


FIGURE 25. Deferred unidirectional traffic.

between them in performance. On our platform, SHA-256 is slightly faster than AES-128 when operating on one block. We note that when implemented in hardware, the circuit size/performance trade-offs are superior for EtUK because an implementation of OTEAE_2 would only have to implement AES.Enc. Moreover, a larger range of CPUs provides hardware acceleration for AES-128 than for SHA-256. Therefore, it is reasonable to conclude that EtUK instantiated with OTEAE_2 fares better on average on most platforms when compared with EtH.

Comparing our SARCAD schemes with liteARCAD, we remark that both EtH and EtUK are more than an order of magnitude faster. Although already quite efficient, liteARCAD has sending/receiving 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. In this case, the number of cryptographic operations per message is high: sending a message requires one hash and $n+1$ symmetric encryptions. Moreover, fresh randomness is required for each encryption. Receiving a message is similar. The ACD protocol [8] is fully based on symmetric primitives during a phase when the direction of communication does not change. Nevertheless, the overall protocol comes with a significant implementation overhead due to the more complex nature of the involved algorithms, effectively rendering it less efficient. This is most evident when the direction of communication constantly changes (Fig. 23) in particular because a new Diffie-Hellman key exchange is required each change.

All the other protocols are based on at least public-key cryptography and perform several orders of magnitudes worse. In particular, the HIBE-based protocols which require expensive pairing operations and JMM all have $O(n^2)$ worst-case complexity (evident in Fig. 25).

7. CONCLUSION

We have shown that the lightweight EtH protocol provides confidentiality and authentication in a strong sense. Namely, it provides forward secrecy and both confidentiality and unforgeability except when there are trivial key exposures. It can be used as a more efficient replacement of liteARCAD in the hybrid ratchet-on-demand protocol with security awareness by Caforio *et al.* [16]. Furthermore, we avoided the use of a random oracle by leveraging a combined security assumption of encryption and hashing (PR security). Alternatively, we give a EtUK protocol based on a newly introduced integrated OTEAE primitive, either based on any secure AEAD or AES-128 and AES-GCM. In many environments, our OTEAE construction based on AES-128 and AES-GCM admits superior performance characteristics.

DATA AVAILABILITY

The data underlying this article are available on Github at https://github.com/dcol97/tcj_sarcad_data.

FUNDING

This work was supported by the National Natural Science Foundation of China (Grant No.62032014, No.61772519).

REFERENCES

[1] Systems, O.W. (2017) Signal protocol library for java/android. https://github.com/WhisperSystems/libsignal-protocol-java. Accessed on November 19, 2019.

[2] Perrin, T. and Marlinspike, M. (2016) *The double ratchet algorithm*. GitHub wiki.
[3] (December 2017). *Whatsapp encryption overview: technical white paper*. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf. Accessed on November 19, 2019.
[4] Lund, J. (2018) Signal partners with Microsoft to bring end-to-end encryption to skype. https://signal.org/blog/skype-partnership/. Accessed on November 19, 2019.
[5] Marlinspike, M. (2016) Open whisper systems partners with google on end-to-end encryption for Allo. https://signal.org/blog/allo.
[6] Cohn-Gordon, K., Cremers, C. and Garratt, L. (2016) On post-compromise security. In *CSF 2016*. Lisboa, Portugal, 27 June - 1 July, pp. 164–178. IEEE, New York.
[7] Bellare, M., Singh, A.C., Jaeger, J., Nyayapati, M. and Stepanovs, I. (2017) Ratcheted encryption and key exchange: The security of messaging. In *CRYPTO 2017*. Santa Barbara, CA, 20–24 August, pp. 619–650. Springer, Cham.
[8] Alwen, J., Coretti, S. and Dodis, Y. (2019) The double ratchet: Security notions, proofs, and modularization for the signal protocol. In *EUROCRYPT 2019*. Darmstadt, Germany, 19–23 May, pp. 129–158. Springer, Cham.
[9] Günther, C.G. (1989) An identity-based key-exchange protocol. In *EUROCRYPT 1989*. Belgium, 10–13 April, pp. 29–37. Springer, Berlin.
[10] Bellare, M. and Yee, B. (2003) Forward-security in private-key cryptography. In San Francisco, C.A. (ed) 13–17 April*Topics in Cryptology — CT-RSA 2003*, pp. 1–18. Springer, Berlin.
[11] Borisov, N., Goldberg, I. and Brewer, E.A. (2004) Off-the-record communication, or, why not to use PGP. In *WPES '04*28 October, pp. 77–84. ACM, New York.
[12] Cohn-Gordon, K., Cremers, C.J.F., Dowling, B., Garratt, L. and Stebila, D. (2017) A formal security analysis of the signal messaging protocol. In *EuroS&P*. Paris, 26–28 April, pp. 451–466. IEEE, New York.
[13] Blazy, O., Bossuat, A., Bultel, X., Fouque, P., Onete, C. and Pagnin, E. (2019) SAID: reshaping signal into an identity-based asynchronous messaging protocol with authenticated ratcheting. In *EuroS&P*. Stockholm, Sweden, 17–19 June, pp. 294–309. IEEE, New York.
[14] Poettering, B. and Rösler, P. (2018) Towards bidirectional ratcheted key exchange. In *CRYPTO 2018*. Santa Barbara, CA, 19–23 August, pp. 3–32. Springer, Cham.
[15] Jaeger, J. and Stepanovs, I. (2018) Optimal channel security against fine-grained state compromise: the safety of messaging. In *CRYPTO 2018*. Santa Barbara, CA, 19–23 August, pp. 33–62. Springer, Cham.
[16] Caforio, A., Durak, F.B. and Vaudenay, S. (2021) Beyond security and efficiency: On-demand ratcheting with security awareness. In *PKC 2021*. Zoom Meeting, 10–13 May, pp. 649–677. Springer, Berlin.
[17] Jost, D., Maurer, U. and Mularczyk, M. (2019) Efficient ratcheting: Almost-optimal guarantees for secure messaging. In Ishai, Y., Rijmen, V. (eds) *EUROCRYPT 2019*. Darmstadt, Germany, 19–23 May, pp. 159–188. Springer, Cham.
[18] Durak, F.B. and Vaudenay, S. (2019) Bidirectional asynchronous ratcheted key agreement with linear complexity. In *IWSEC 2019*. Tokyo, Japan, 28–30 August, pp. 343–362. Springer, Cham.
[19] Jost, D., Maurer, U. and Mularczyk, M. (2019) A unified and composable take on ratcheting. In *TCC 2019*. Nuremberg, Germany, 1-5 December, pp. 180–210. Springer, Cham.
[20] Balli, F., Rösler, P. and Vaudenay, S. (2020) Determining the core primitive for optimally secure ratcheting. In *ASIACRYPT 2020*. Daejeon, South Korea, 7-11 December, pp. 621–650. Springer, Cham.
[21] Yan, H. and Vaudenay, S. (2020) Symmetric asynchronous ratcheted communication with associated data. In *IWSEC 2020*. Fukui, Japan, 2-4 September, pp. 184–204. Springer, Cham.
[22] Rogaway, P. (2002) Authenticated-encryption with associated-data. In *CCS 2002*. Washington, DC, 18–22 November, pp. 98–107. ACM, New York.
[23] McGrew, D. and Viega, J. (2004) The Galois/counter mode of operation (GCM). In *Submission to NIST Modes of Operation Process*. NIST, US.
[24] Dang, Q. (2015) Secure hash standard. Federal Inf. In *Process. Stds. (NIST FIPS)*. National Institute of Standards and Technology, Gaithersburg, MD.
[25] Daemen, J. and Rijmen, V. (2002) *The design of Rijndael*. Springer, Berlin.
[26] Dworkin, M.J. (2007) *Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC*. National Institute of Standards & Technology.
[27] McGrew, D.A. and Viega, J. (2004) The security and performance of the Galois/counter mode (GCM) of operation. In *INDOCRYPT 2004*. Chennai, India, 20–22 December, pp. 343–355. Springer, Berlin.
[28] Iwata, T., Ohashi, K. and Minematsu, K. (2012) Breaking and repairing GCM security proofs. In *CRYPTO 2012*Santa Barbara, CA, 19–23 August, pp. 31–49. Springer, Berlin.
[29] Bellare, M. and Rogaway, P. (2006) The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT 2006*. St. Petersburg, Russia, 28 May - 1 June, pp. 409–426. Springer, Berlin.

A. LITEARCAD PROTOCOL

In [16], Caforio et al. proposed a generic construction of a messaging protocol offering *on-demand ratcheting*, by composing a strongly secure protocol, such as DV protocol [18], and a weakly secure but lighter one, such as liteARCAD. This hybrid system allows the sender to select which security level he wants to use. We recall the liteARCAD protocol in Fig. A.1. Note that Send is randomized and that a failed Receive call sets the caller's state to $\perp$, unlike our SARCAD schemes.

B. AES-GCM

We describe AES-GCM [23] instantiated with the block cipher AES-128 [25]; our description is based on that of Iwata *et al.* [28]. Let $\mathsf{str}_n(x)$ be a deterministic injective function that encodes a positive integer $x < 2^{64} - 1$ as an n-bit string. Let $\mathsf{msb}_n(X)$ be the algorithm that extracts the n most significant bits of bitstring X (for our use we will have $|X| \geq n$). Let '$\cdot$' denote multiplication modulo a field (possibly of characteristic 0) which is deduced from context, and '$\oplus$' denote bitwise XOR. We present AES-GCM in Fig. B.1. Note that we assume that $|\mathsf{nc}| = 96$, which slightly simplifies the algorithm description, and that the tag output by AES-GCM.Enc is 128 bits long.

liteARCAD.Setup $= \mathrm{H.Gen}$

liteARCAD.Initall$(\mathsf{pp} = (1^\lambda, \mathsf{hk}))$

1: pick $\mathsf{sk}_1, \mathsf{sk}_2 \xleftarrow{\$} \mathrm{OTAE}.\mathcal{K}(\lambda)$
2: $\mathsf{st}_A^{\mathsf{send}} \leftarrow (\lambda, \mathsf{hk}, (\mathsf{sk}_1), (\mathsf{sk}_2))$
3: $\mathsf{st}_B^{\mathsf{send}} \leftarrow (\lambda, \mathsf{hk}, (\mathsf{sk}_2), (\mathsf{sk}_1))$
4: **return** $(\mathsf{st}_A, \mathsf{st}_B, \perp)$

uni.Init(1^λ)

1: pick $\mathsf{sk} \xleftarrow{\$} \mathrm{OTAE}.\mathcal{K}(\lambda)$
2: $\mathsf{st}_S \leftarrow \mathsf{sk}$
3: $\mathsf{st}_R \leftarrow \mathsf{sk}$
4: **return** $(\mathsf{st}_S, \mathsf{st}_R)$

uni.Send$(1^\lambda, \mathsf{hk}, \vec{\mathsf{st}}_S, \mathsf{ad}, \mathsf{pt})$

1: pick $\mathsf{sk} \xleftarrow{\$} \mathrm{OTAE}.\mathcal{K}(\lambda)$
2: $\mathsf{pt}' \leftarrow (\mathsf{sk}, \mathsf{pt})$
3: $\mathrm{onion.Enc}(1^\lambda, \mathsf{hk}, \vec{\mathsf{st}}_S, \mathsf{ad}, \mathsf{pt}') \to \vec{\mathsf{ct}}$
4: **return** $(\mathsf{sk}, \vec{\mathsf{ct}})$

uni.Receive$(\mathsf{hk}, \vec{\mathsf{st}}_R, \mathsf{ad}, \vec{\mathsf{ct}})$

1: $\mathrm{onion.Dec}(\mathsf{hk}, \vec{\mathsf{st}}_R, \mathsf{ad}, \vec{\mathsf{ct}}) \to \mathsf{pt}'$
2: **if** $\mathsf{pt}' = \perp$ **then**
3: **return** $(\mathsf{false}, \perp, \perp)$
4: parse $\mathsf{pt}' = (\mathsf{sk}, \mathsf{pt})$
5: **return** $(\mathsf{true}, \mathsf{sk}, \mathsf{pt})$

onion.Send$(1^\lambda, \mathsf{hk}, \mathsf{st}_S^1, \ldots, \mathsf{st}_S^n, \mathsf{ad}, \mathsf{pt})$

1: pick $\mathsf{k}_1, \ldots, \mathsf{k}_n \xleftarrow{\$} \{0,1\}^{\mathrm{Sym.kl}(\lambda)}$
2: $\mathsf{k} \leftarrow \mathsf{k}_1 \oplus \cdots \oplus \mathsf{k}_n$
3: $\mathsf{ct}_{n+1} \leftarrow \mathrm{Sym.Enc}(\mathsf{k}, \mathsf{pt})$
4: $\mathsf{ad}_{n+1} \leftarrow \mathsf{ad}$
5: **for** $i = n$ down to 1 **do**
6: $\quad \mathsf{ad}_i \leftarrow \mathrm{H.Eval}(\mathsf{hk}, \mathsf{ad}_{i+1}, n, \mathsf{ct}_{i+1})$
7: $\quad \mathsf{ct}_i \leftarrow \mathrm{OTAE.Enc}(\mathsf{st}_S^i, \mathsf{ad}_i, \mathsf{k}_i)$
8: **return** $(\mathsf{ct}_1, \ldots, \mathsf{ct}_{n+1})$

onion.Receive$(\mathsf{hk}, \mathsf{st}_R^1, \ldots, \mathsf{st}_R^n, \mathsf{ad}, \vec{\mathsf{ct}})$

1: parse $\vec{\mathsf{ct}} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_{n+1})$
2: $\mathsf{ad}_{n+1} \leftarrow \mathsf{ad}$
3: **for** $i = n$ down to 1 **do**
4: $\quad \mathsf{ad}_i \leftarrow \mathrm{H.Eval}(\mathsf{hk}, \mathsf{ad}_{i+1}, n, \mathsf{ct}_{i+1})$
5: $\quad \mathrm{OTAE.Dec}(\mathsf{st}_R^i, \mathsf{ad}_i, \mathsf{ct}_i) \to \mathsf{k}_i$
6: $\quad$ **if** $\mathsf{k}_i = \perp$ **then**
7: $\quad\quad$ **return** $\perp$
8: $\mathsf{k} \leftarrow \mathsf{k}_1 \oplus \cdots \oplus \mathsf{k}_n$
9: $\mathsf{pt} \leftarrow \mathrm{Sym.Dec}(\mathsf{k}, \mathsf{ct}_0)$
10: **return** pt

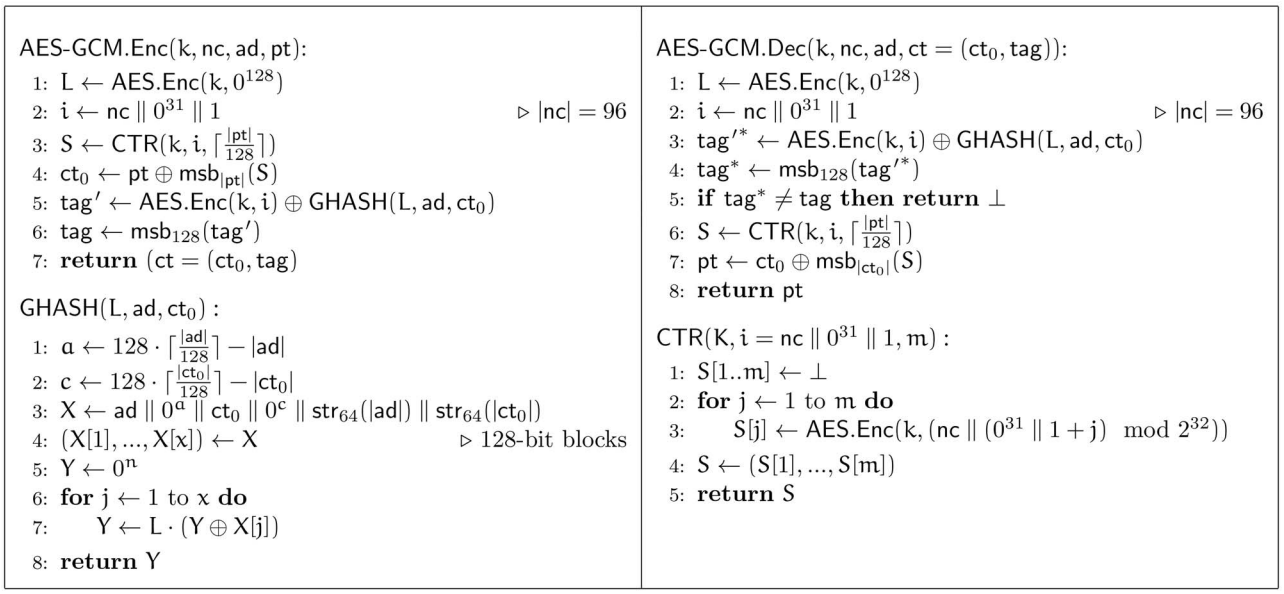
liteARCAD.Send$(\mathsf{st}_P, \mathsf{ad}, \mathsf{pt})$

1: parse $\mathsf{st}_P = (\lambda, \mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v}))$
2: $\mathrm{uni.Init}(1^\lambda) \xrightarrow{\$} (\mathsf{st}_{\mathsf{Snew}}, \mathsf{st}_P^{\mathsf{rec},v+1})$ ▷ append a new receive state to the $\mathsf{st}_P^{\mathsf{rec}}$ list
3: $\mathsf{pt}' \leftarrow (\mathsf{st}_{\mathsf{Snew}}, \mathsf{pt})$ ▷ then, $\mathsf{st}_{\mathsf{Snew}}$ is erased to avoid leaking
4: take the smallest i s.t. $\mathsf{st}_P^{\mathsf{send},i} \neq \perp$ ▷ $i = u - n$ if we had n Receive since the last Send
5: $\mathrm{uni.Send}(1^\lambda, \mathsf{hk}, \mathsf{st}_P^{\mathsf{send},i}, \ldots, \mathsf{st}_P^{\mathsf{send},u}, \mathsf{ad}, \mathsf{pt}') \xrightarrow{\$} (\mathsf{st}_P^{\mathsf{send},u}, \mathsf{ct})$ ▷ update $\mathsf{st}_P^{\mathsf{send},u}$
6: $\mathsf{st}_P^{\mathsf{send},i}, \ldots, \mathsf{st}_P^{\mathsf{send},u-1} \leftarrow \perp$ ▷ flush the send state list: only $\mathsf{st}_P^{\mathsf{send},u}$ remains
7: $\mathsf{st}'_P \leftarrow (\lambda, \mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v+1}))$
8: **return** $(\mathsf{st}'_P, \mathsf{ct})$

liteARCAD.Receive$(\mathsf{st}_P, \mathsf{ad}, \mathsf{ct})$

9: parse $\mathsf{st}_P = (\lambda, \mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v}))$
10: set $n+1$ to the number of components in ct ▷ the onion has n layers
11: set i to the smallest index such that $\mathsf{st}_P^{\mathsf{rec},i} \neq \perp$
12: **if** $i + n - 1 > v$ **then return** $(\mathsf{false}, \mathsf{st}_P, \perp)$
13: $\mathrm{uni.Receive}(\mathsf{hk}, \mathsf{st}_P^{\mathsf{rec},i}, \ldots, \mathsf{st}_P^{\mathsf{rec},i+n-1}, \mathsf{ad}, \mathsf{ct}) \to (\mathsf{acc}, \mathsf{st}'^{\mathsf{rec},i+n-1}_P, \mathsf{pt}')$
14: **if** $\mathsf{acc} = \mathsf{false}$ **then return** $(\mathsf{false}, \mathsf{st}_P, \perp)$
15: parse $\mathsf{pt}' = (\mathsf{st}_P^{\mathsf{send},u+1}, \mathsf{pt})$ ▷ a new send state is added in the list
16: $\mathsf{st}_P^{\mathsf{rec},i}, \ldots, \mathsf{st}_P^{\mathsf{rec},i+n-2} \leftarrow \perp$ ▷ update stage 1: $n-1$ entries of $\mathsf{st}_P^{\mathsf{rec}}$ were erased
17: $\mathsf{st}_P^{\mathsf{rec},i+n-1} \leftarrow \mathsf{st}'^{\mathsf{rec},i+n-1}_P$ ▷ update stage 2: update $\mathsf{st}_P^{\mathsf{rec},i+n-1}$
18: $\mathsf{st}'_P \leftarrow (\lambda, \mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u+1}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v}))$
19: **return** $(\mathsf{acc}, \mathsf{st}'_P, \mathsf{pt})$

FIGURE A.1. liteARCAD protocol.



AES-GCM.Enc(k, nc, ad, pt):
1: $L \leftarrow \mathsf{AES.Enc}(k, 0^{128})$
2: $i \leftarrow nc \,||\, 0^{31} \,||\, 1$ ▷ $|nc| = 96$
3: $S \leftarrow \mathsf{CTR}(k, i, \lceil \frac{|pt|}{128} \rceil)$
4: $ct_0 \leftarrow pt \oplus \mathsf{msb}_{|pt|}(S)$
5: $tag' \leftarrow \mathsf{AES.Enc}(k, i) \oplus \mathsf{GHASH}(L, ad, ct_0)$
6: $tag \leftarrow \mathsf{msb}_{128}(tag')$
7: **return** $(ct = (ct_0, tag)$

GHASH(L, ad, ct_0) :
1: $a \leftarrow 128 \cdot \lceil \frac{|ad|}{128} \rceil - |ad|$
2: $c \leftarrow 128 \cdot \lceil \frac{|ct_0|}{128} \rceil - |ct_0|$
3: $X \leftarrow ad \,||\, 0^a \,||\, ct_0 \,||\, 0^c \,||\, \mathsf{str}_{64}(|ad|) \,||\, \mathsf{str}_{64}(|ct_0|)$
4: $(X[1], ..., X[x]) \leftarrow X$ ▷ 128-bit blocks
5: $Y \leftarrow 0^n$
6: **for** $j \leftarrow 1$ to x **do**
7: $\quad Y \leftarrow L \cdot (Y \oplus X[j])$
8: **return** Y

AES-GCM.Dec(k, nc, ad, ct $= (ct_0, tag)$):
1: $L \leftarrow \mathsf{AES.Enc}(k, 0^{128})$
2: $i \leftarrow nc \,||\, 0^{31} \,||\, 1$ ▷ $|nc| = 96$
3: $tag'^{*} \leftarrow \mathsf{AES.Enc}(k, i) \oplus \mathsf{GHASH}(L, ad, ct_0)$
4: $tag^{*} \leftarrow \mathsf{msb}_{128}(tag'^{*})$
5: **if** $tag^{*} \neq tag$ **then return** $\perp$
6: $S \leftarrow \mathsf{CTR}(k, i, \lceil \frac{|pt|}{128} \rceil)$
7: $pt \leftarrow ct_0 \oplus \mathsf{msb}_{|ct_0|}(S)$
8: **return** pt

CTR$(K, i = nc \,||\, 0^{31} \,||\, 1, m)$:
1: $S[1..m] \leftarrow \perp$
2: **for** $j \leftarrow 1$ to m **do**
3: $\quad S[j] \leftarrow \mathsf{AES.Enc}(k, (nc \,||\, (0^{31} \,||\, 1 + j) \mod 2^{32}))$
4: $S \leftarrow (S[1], ..., S[m])$
5: **return** S

FIGURE B.1. AES-GCM instantiated with block cipher AES-128 (AES) with 128-bit keys, 128-bit tags and 96-bit nonces.