Privacy-Preserving Authenticated Key Exchange for Constrained Devices*

Loïc Ferreira

Orange Labs, Applied Crypto Group, Caen, France
loic.ferreira@orange.com

Abstract. In this paper we investigate the field of privacy-preserving authenticated key exchange protocols (PPAKE). First we make a cryptographic analysis of a previous PPAKE protocol. We show that most of its security properties, including privacy, are broken, despite the security proofs that are provided. Then we describe a strong security model which captures the security properties of a PPAKE: entity authentication, key indistinguishability, forward secrecy, and privacy. Finally, we present a PPAKE protocol in the symmetric-key setting which is suitable for constrained devices. We formally prove the security of this protocol in our model.

Keywords: Authenticated key agreement · Internet of Things · Cryptanalysis · Privacy · PPAKE · Security model.

1 Introduction

Entity authentication and indistinguishability for the session key are the primary goals that a key exchange protocol aims at achieving. With the growth of social networks, and virtual communications, privacy-preserving techniques have gained interest in the design of real-world protocols (e.g., TLS 1.3 [32]). With the development of the Internet of Things (IoT) and its novel use cases interest in privacy is revived.

IoT provides applications in many fields: patient remote monitoring, energy consumption, air pollution control, traffic management, retail and logistics, etc. IoT technologies deal with and combine sets of data which makes increasingly difficult to distinguish between information that enable identification and information which do not [38]. For instance, smartphones gather critical amount of private data about their owner (identifiers, location, activity) that bear privacy risks. The diversity of connected objects form a large intelligent network that can serve as a medium for the leakage of personal data [29]. Rather soon the threats induced by the distributed nature of the IoT have been highlighted [23], among which one can cite identification, tracking, and profiling.

* A preliminary version of this paper appears in the proceedings of the *20th International Conference on Applied Cryptography and Network Security* (ACNS 2022). This is the full version.

Devising a security protocol for the IoT is a challenging task since the devices that must implement and execute the protocol are constrained in terms of energy, computation, and memory in particular. Consequently, the protocols are often built on symmetric-key functions for efficiency reasons. In turn, these "symmetric" protocols do not achieve the same security properties as "asymmetric" protocols (i.e., based on public-key schemes). Adding yet another security property (privacy) is not a trivial task.

In this paper we focus our attention on the SAKE protocol proposed by Avoine, Canard, and Ferreira [5]. Built solely upon symmetric-key functions, SAKE is an efficient protocol for constrained devices. It provides mutual authentication, key exchange, and forward secrecy. Its security is proved in a strong model (roughly the same type of model as those used to analyse protocols based on asymmetric algorithms). Moreover, with a suitable choice of symmetric functions, SAKE is quantum secure. This raised the attention of the French National Cybersecurity Agency (ANSSI) which indicates that SAKE is a possible alternative to current "classical" authenticated key exchange (AKE) protocols in a quantum world [2]. Our goal is to turn SAKE into a privacy-preserving protocol, while keeping all its security properties, and to formally prove the security of the resulting protocol in a model at least as strong as that of used to analyse the original protocol.

1.1 Related Work

Most of the privacy-preserving protocols for low-resource devices, and the corresponding adversarial models are related to the RFID field (e.g., [7, 17–19, 25, 27, 28, 36] to cite a few). Privacy-preserving mechanisms have also been investigated in other IoT contexts such as smart homes [31, 35] or low-power wide area networks (LPWAN) [4, 37]. However most of these works consider the privacy property only, focus on a specific setting (LPWAN), require a specific hardware (physically unclonable functions), or are built on questionable techniques with respect to security and efficiency (chaotic maps).

In [1], Aghili, Jolfaei, and Abidin propose a privacy-preserving authenticated key exchange protocol (PPAKE) with forward secrecy dedicated to IoT. This protocol builds upon Avoine et al.'s proposal [5]. Aghili et al. propose a variant of this protocol that aims in particular at guaranteeing privacy. However, and despite the security proofs they provide, their proposal is flawed (see Section 3).

Restarting from Avoine et al.'s protocol, we fix the issues of Aghili et al.'s protocol, and devise a clean and proper security model that we use to formally prove the security of the corrected PPAKE protocol.

1.2 Contributions

In this paper we investigate the field of privacy-preserving authenticated protocols. First we make a security analysis of a previous PPAKE protocol [1]. Then

we describe a new security model which captures privacy (among other security properties) for authenticated key exchange protocols. Finally, we present a PPAKE protocol secure in our model.

Cryptanalysis. In [1], Aghili et al. propose a PPAKE protocol dedicated to IoT. Built upon a previous work by Avoine et al. [5], their proposal aims at keeping the same security properties as Avoine et al.'s protocol: entity authentication, key indistinguishability, and forward secrecy, and at being resistant to several attacks: "replay attacks", "time-based attack", and "tracking" (cf. [1, Sections 6.1 and 9]). We make a cryptographic analysis of Aghili et al.'s protocol and show that most of the claimed security properties are broken (we respect the same attack settings that are considered in [1], in particular the powers granted to the adversary).

Security Model. We present a security model that captures strong guarantees for authenticated key exchange protocols. We extend the security model used by Avoine et al. [5] to prove the security of their protocol by introducing a criterion for indistinguishability of identities. That is, in order to define the privacy property, we borrow the concept of *virtual identifier* from Hermans, Pashalidis, Vercauteren, and Preneel [24], which appears also in Ouafi and Phan [30]. This concept allows hiding the identity of the party the adversary interacts with. The privacy property guarantees not only that the identity of an end-device is hidden, but that two different protocol runs are *unlinkable*. We also follow the paradigm proposed by Schwenk, Schäge, and Lauer [33], and incorporate the privacy property together with the other security properties. This approach guarantees that the different security properties are independent of each other. More specifically, our resulting model requires that, say, the key indistinguishability property holds even in the presence of attacks that adaptively unmask identities. Conversely, confidentiality of identities is ensured even in the presence of queries that let the attacker reveal session keys. This yields a strong security model which can serve as a tool to analyse other authenticated key exchange protocols that implement mechanisms to guarantee privacy.

Privacy-Preserving AKE. Starting anew from the SAKE protocol proposed by Avoine et al., we take another look at the concept of PPAKE for constrained devices. To the security properties guaranteed by SAKE, we add privacy. This results in a PPAKE protocol suitable for constrained devices that we naturally call Privacy-Preserving SAKE (PPSAKE). We formally prove that PPSAKE is secure in our strong security model.

2 Description of the SAKE Protocol

2.1 SAKE

SAKE [5] is a two-party AKE based on symmetric-key functions and pre-shared keys (see Figure 2). The two parties A and B share a derivation master key K

and an authentication master key K'. In order to mutually authenticate, each party exchanges a pseudo-random value (r_A, r_B). A MAC tag is computed over this challenge and returned to the sender (messages m_B and m_A). The session key is computed from the two pseudo-random values r_A and r_B and the derivation master key: $sk \leftarrow \mathsf{KDF}(K, r_A, r_B)$. Forward secrecy is guaranteed by using a key evolving scheme. That is, once both parties are mutually authenticated and the session key is computed, the derivation master key is updated with a one-way function: $K \leftarrow \mathsf{update}(K)$. Therefore the previous session keys remain safe even if (updated) K is disclosed.

As soon as two parties make a shared (symmetric) key evolve, a synchronisation problem arises: one of the parties has to make the first move whereas the other remains late, at least temporarily. This issue is solved with the authentication master key. The initiator A in SAKE stores the authentication keys corresponding to three consecutive epochs: previous (K'_{j-1}), current (K'_j), and future (K'_{j+1}) (see Figure 1).

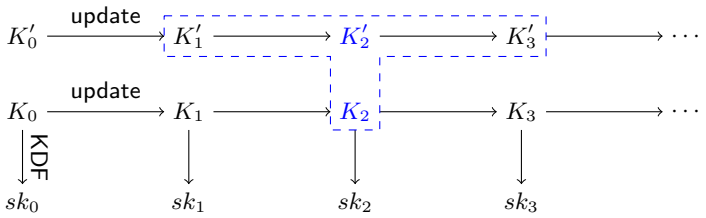

Fig. 1: Party A stores authentication master keys corresponding to three consecutive epochs ($j-1$, j, $j+1$), and one derivation master key (illustration with $j=2$ with the blue dashed box). Party B stores one sample of each master key (in blue).

Upon reception of the MAC-ed challenge computed by B (message m_B), A detects which epoch B belongs to by checking its MAC tag. Then, in the subsequent message (m_A), A indicates B if it must catch up (with the bit ϵ). Likewise, if A is late, it updates its master keys and then proceeds with the regular operations (upon reception of message τ'_B). Eventually, both parties update the authentication master keys the same way they do for the derivation master key. Only the initiator needs to keep the authentication master keys of three consecutive epochs. Avoine et al. have proved that the initiator A can only be either *one step* behind, or in sync, or *one step* ahead to B (hence the figure of three keys K'_{j-1}, K'_j, K'_{j+1}). That is $\delta_{AB} \in \{-1, 0, 1\}$ where δ_{AB} is the gap between A and B. Since the derivation master key and the authentication master key are independent, keeping previous authentication master keys does not jeopardise forward secrecy.

Once a correct and complete session ends, three goals are achieved in the same protocol run: (i) the two parties have updated their master keys, (ii) their master

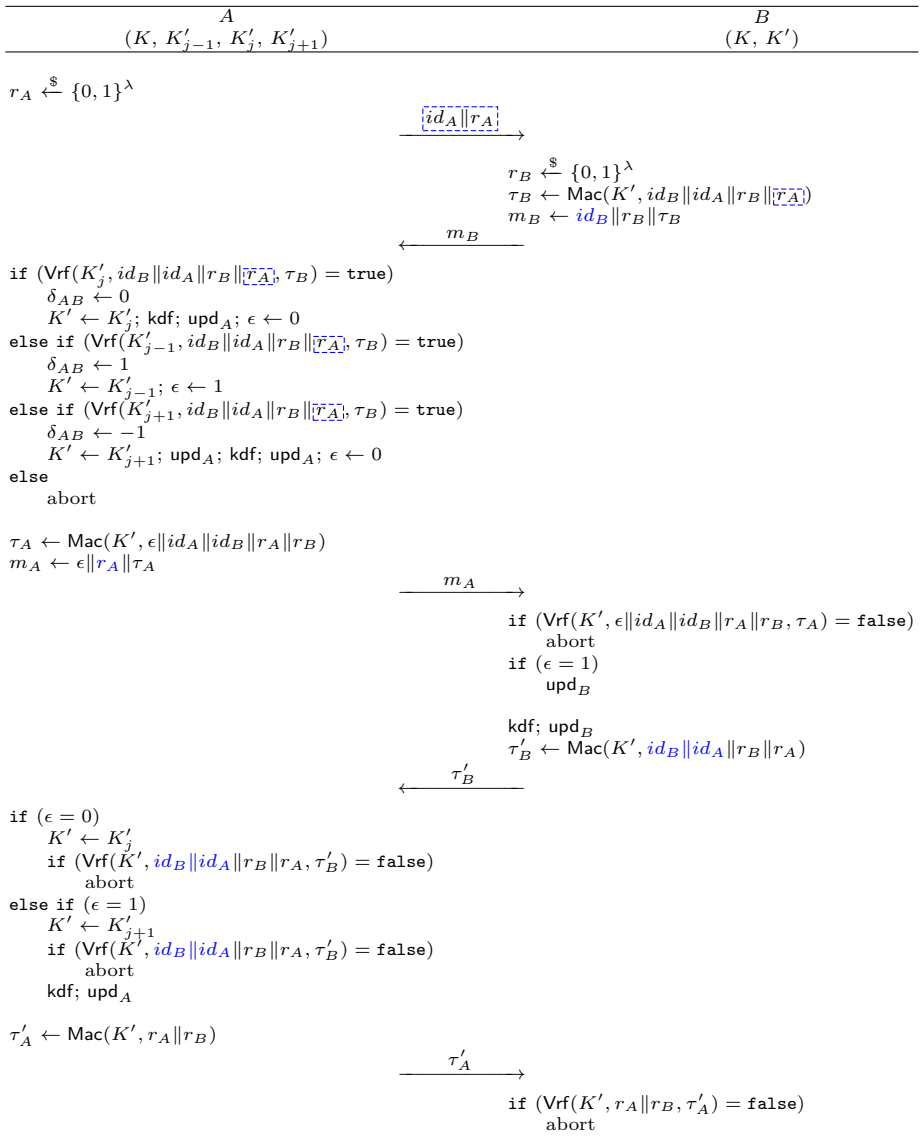

Fig. 2: The SAKE/SAKE-AM protocol. Elements surrounded by a blue dashed box appear only in SAKE. Elements in blue appear only in SAKE-AM.

keys are synchronised, and (iii) they share a new session key. Therefore mutual authentication, key exchange (with forward secrecy), and resynchronisation are done in the continuity of a single session. Moreover, there is no need for an additional procedure (e.g., resynchronisation phase) or functionality (e.g., shared clock). The protocol is made of five messages at most, and can be reduced to four messages if the two parties are synchronised at the beginning of the session.

Notations. For the sake of clarity, we use the following notation in Figure 2:

- kdf corresponds to: $sk \leftarrow \mathsf{KDF}(K, r_A, r_B)$
- upd_A corresponds to
  1. $K \leftarrow \mathsf{update}(K)$
  2. $K'_{j-1} \leftarrow K'_j$
  3. $K'_j \leftarrow K'_{j+1}$
  4. $K'_{j+1} \leftarrow \mathsf{update}(K'_{j+1})$
- upd_B corresponds to
  1. $K \leftarrow \mathsf{update}(K)$
  2. $K' \leftarrow \mathsf{update}(K')$

Moreover, $\mathsf{Vrf}(k, m, \tau)$ denotes the MAC verification function that takes as input a secret key k, a message m, and a tag τ. It outputs `true` if τ is a valid tag on message m with respect to k. Otherwise, it returns `false`.

2.2 SAKE-AM

From SAKE a complementary mode can be derived: SAKE-AM (which stands for "agressive mode"). Compared to SAKE, the first message ($id_A \| r_A$) is skipped. Hence, in SAKE-AM, B is the initiator (and stores two master keys K, K'). What becomes the first message is computed as $m_B = id_B \| r_B \| \tau_B$ with $\tau_B = \mathsf{Mac}(K', id_B \| id_A \| r_B)$. The second message is computed as $m_A = \epsilon \| r_A \| \tau_A$ (with τ_A computed as in SAKE). The other messages and calculations are essentially the same as in SAKE.

Used together, SAKE and SAKE-AM allow any party to be either initiator or responder in a protocol run. Moreover the smallest amount of calculation is always done by the same party (irrespective of its role). This is particularly convenient in the context of a set of end-devices communicating with a back-end server. When the end-device wants to initiate a communication, protocol SAKE-AM is launched. Otherwise (the server is initiator), SAKE is used. Therefore, the end-device always does the lightest computations.

3 A Flawed Proposal

In [1] Aghili et al. propose to modify SAKE/SAKE-AM in order to turn the protocol into a privacy-preserving scheme.[1] They consider a setting where party

[1] Our description of Aghili et al.'s protocols is mainly based on Section 6 complemented with Section C.2 of their paper [1].

A is a server communicating with a set of end-devices (many parties B). They modify the SAKE and SAKE-AM protocols in order to achieve three main goals:

1. Forbidding identification and tracking of a party B (in particular with id_B).
2. Forbidding the replay of the first message (m_B) in SAKE-AM. In SAKE-AM, a message m_B corresponding to the previous epoch (i.e., computed with the authentication master key K'_{j-1}) can be replayed multiple times to A (until a new session is completed), and A computes and responds with a message m_A. Although this is not sufficient for the responder A to "accept" and to authenticate the initiator B (eventually the session aborts), Aghili et al. aim at preventing such a possibility. In contrast, in TLS 1.3 with 0-RTT mode [32] the server must deem the initial message (Client Hello) as authentic, and execute the request herein included [21]. Consequently, mitigations are necessary in TLS 1.3 (cf. [32, Section 8]).
3. Forbidding recognition of a party B based on the amount of calculations done by A. In some cases (see below), when party A receives a message m_B in Aghili et al.'s version of SAKE and SAKE-AM, A must try all authentication master keys it stores in its database in order to verify the message, until a match is found. Therefore the time spent by A to find the correct key allows an adversary to recognise which party B communicates with A (the measurement done by the adversary is used as an index that designates B).

3.1 Issues

Aghili et al. modify the SAKE/SAKE-AM protocol as follows (see Figures 8 and 9 in Section B). First they add identifiers in the messages in order for two communicating parties A and B to distinguish which messages are intended to them, among the flow of messages sent by all parties. That is, they necessarily mix the communication and the application (cryptographic) layers since the former may include parameters (identifier) that contradict the goals they want to achieve.[2] In addition, upon reception of m_A, id_B is updated by B (A does also the same): $id_B \leftarrow \mathsf{update}(id_B \| K')$. This new identifier value is transmitted in the subsequent message sent by B and also in the first message of the next protocol run. We can see that there is a first issue since the same identifier id_B is used in two consecutive sessions. Therefore it is trivial to track party B (this contradicts goal 1).

Moreover, id_B is replaced with a pseudo-random value r_α in m_B if message m_A was not received by B during the previous session. The purpose is to avoid that the same identifier id_B be used in two consecutive messages m_B (a correct message m_A triggers the update of id_B, hence id_B remains the same in the absence of such a message). In Aghili et al.'s version of SAKE-AM, this means that $m_B = x \| r_B \| \tau_B$ with $x \in \{id_B, r_\alpha\}$ and $\tau_B = \mathsf{Mac}(K', id_B \| id_A \| r_B)$. When $x = r_\alpha$, party A tries all the authentication master keys K' (corresponding to

[2] In [5], Avoine et al. describe the message flow of a cryptographic protocol. Consequently, they indicate only the parameters that are necessary on a cryptographic point of view.

different communicating parties B) it stores in its database until a match is found. The issue here is that r_α is not included in the computation of τ_B even when it replaces id_B in m_B. Therefore an adversary can alter m_B without A being able to notice the change. This breaks entity authentication because, in the adopted security model, partnership is based on the notion of "matching conversations" (i.e., equality of transcript of messages) [26].

Furthermore this invalidates goal 3. Indeed when the adversary modifies id_B in m_B this compels A to try all the authentication master keys, which helps the adversary to recognise which party B has sent the message m_B, hence to track that party (defeating goal 1 again).

In order to achieve goal 2 (which concerns SAKE-AM only), A stores the pseudo-random value r_B received in the *two previous* sessions. This countermeasure is not enough and can easily be bypassed. The adversary merely intercepts three times consecutively an initial message m_B sent by the initiator B, and not received by A (dropped by the adversary). Next the adversary lets A and B complete successfully one protocol run. The three messages m_B highly likely carry pairwise distinct values r_B. When the adversary sends any of these messages, they are accepted by A because they carry an unknown value r_B, and because they all correspond to the previous epoch from A's perspective (i.e., computed with K'_{j-1}). Therefore A computes and sends a message m_A. Alternatively sending these three messages "flushes" A's memory of r_B values. Hence A keeps sending messages m_A in response.

The issues raised above break the security properties claimed in [1, Sections 6.1 and 9] (respecting the same security experiments and adversarial model considered by Aghili et al.): replay, time-based attack, tracking, entity authentication.

3.2 Countermeasures

The vulnerabilities in Aghili et al.'s proposal can be thwarted as follows. In order to fix the issue in the entity authentication, the pseudo-random value r_α must also be involved in the computation of the MAC tag τ_B of message m_B. In addition the identity parameter must be involved in the computation of the two MAC tags of the messages m'_A and m'_B.

To thwart the replay attack, A must detect *all* values r_B previously received. This can be done efficiently with a Bloom filter [14] or a Cuckoo filter [20]. Such mechanisms allow detecting all replays (no false negative) with a constant storage in memory. The rate of false positives depends on the size of the filter.

The time-based attack can be mitigated by equalising the time spent to explore the set of authentication keys (e.g., all keys are tried even when the correct one is found), or by randomly exploring this set [6].

To forbid any tracking, a value id_B must be used once only per session. That is, an updated version of id_B must be used during each new session.

The vulnerabilities we describe question also the correctness of the security proofs provided in [1], made in the computational model (using the game-based methodology [11,34]), and with the ProVerif verification tool [13]. In particular,

the privacy property is not captured by the security model used in [1]. This highlights the importance of devising and using a suitable security model.

The relevance of the countermeasures is shown in the security proofs (cf. Section 6) for our privacy-preserving AKE protocol described in Section 5.

4 Security Model

In this section, we present our security model for PPAKE protocols. We use the model for authenticated key exchange protocols described by Avoine et al. [5] to prove the security of their SAKE and SAKE-AM protocols, which is based on the model of Brzuska, Jacobsen, and Stebila [16]. This model captures entity authentication, key indistinguishability, and forward secrecy in the symmetric-key setting.

We extend this model by introducing a criterion for indistinguishability of identities. That is, in order to define the privacy property, we borrow the concept of virtual identifier from Ouafi at al. [30] and Hermans et al. [24]. This concept allows hiding the identity of the party the adversary is interacting with. The privacy property guarantees not only that the identity of an end-device is hidden, but that two different protocol runs are *unlinkable*. Given the two-party protocol which we want to prove the security, and its deployment context, we aim at guaranteeing the end-device's privacy only. However our model can be extended in a straightforward manner to provide privacy to any party involved in a protocol run (end-device and server).

Finally, we follow the paradigm proposed by Schwenk et al. [33], and incorporate the privacy property together with the other security properties. This approach guarantees that the different security properties are independent of each other. More specifically, our resulting model requires that, say, the key indistinguishability property holds even in the presence of attacks that adaptively unmask identities. Conversely, confidentiality of identities is ensured even in the presence of queries that let the attacker reveal session keys. Hence our model is stronger than models where security properties are considered separately (e.g., privacy and key indistinguishability), and not all the adversarial queries are available in all the security experiments (e.g. [3, 22]).

In our model, the long-term symmetric keys shared by the two communicating parties can not be given to the adversary before the session is completed (i.e., our security model does not capture key compromise impersonation attacks [14]). However these keys can be disclosed once one of the two instances accepts (this captures forward secrecy). This makes our model stronger than other models used in the symmetric-key setting (e.g., [22]), and comparable in terms of powers granted to the adversary to security models used in the asymmetric setting (e.g., [16, 26]).

We do think that this security model can serve as a tool to analyse other authenticated key exchange protocols that implement mechanisms to guarantee privacy.

4.1 Execution Environment

Parties. Let $\mathcal{E}$ be a set of end-devices, and $\mathcal{S}$ a set of servers. The type of a party P_i is denoted $\mathsf{type}(P_i) \in \{\mathsf{end\text{-}device}, \mathsf{server}\}$.

A two-party protocol is carried out by an end-device and a server. Each party $P_i \in \mathcal{E} \cup \mathcal{S}$ has an associated long-term key $P_i.\mathsf{ltk}$, and is identified with two parameters: its permanent identifier which we also denote by P_i, and its current identifier $P_i.\mathsf{id}$. The same long-term key is shared by a unique pair of parties (P_i, P_j). That is: $P_i.\mathsf{ltk} = P_j.\mathsf{ltk}$.

In addition, a party $P_i \in \mathcal{S}$ stores a database which each entry corresponds to the long-term key of an $\mathsf{end\text{-}device}$ party $P_j.\mathsf{ltk}$, its current identifier $P_j.\mathsf{id}$, along with its permanent identifier P_j.

Instances. Each party can take part in multiple sequential executions of the protocol. We prohibit *parallel* executions of the protocol. Indeed, since the protocol we propose is based on shared *evolving* symmetric keys, running multiple instances in parallel may cause some executions to abort.[3] This is the only restriction we demand compared to AKE security models used in the public-key setting.

Each run of the protocol is called a session. To each session of a party P_i, an instance π_i^s is associated which embodies this (local) session's execution of the protocol, and has access to the long-term key and current identifier of the party. P_i is called the *parent* of π_i^s, and the type of an instance is the type of its parent: $\mathsf{type}(\pi_i^s) = \mathsf{type}(P_i) \in \{\mathsf{end\text{-}device}, \mathsf{server}\}$. In addition, each instance maintains the following state specific to the session.

- ρ: the role $\rho \in \{\mathsf{initiator}, \mathsf{responder}\}$ of the session in the protocol execution, being either the initiator or the responder.
- pid: the identity $\mathsf{pid} \in \mathcal{P}$ of the intended communication partner of π_i^s.
- α: the state $\alpha \in \{\perp, \texttt{running}, \texttt{accepted}, \texttt{rejected}\}$ of the instance.
- sk: the session key derived by π_i^s.
- κ: the status $\kappa \in \{\perp, \texttt{revealed}\}$ of the session key $\pi_i^s.\mathsf{sk}$.
- sid: the identifier of the session.
- b: a random bit $\mathsf{b} \in \{0, 1\}$ sampled at initialisation of π_i^s.

We use the notion of $\mathsf{initiator}$ and $\mathsf{responder}$ on the one hand, and $\mathsf{end\text{-}device}$, server on the other hand. An $\mathsf{initiator}$ instance sends the first message of the protocol, whereas a $\mathsf{responder}$ instance responds to it. An $\mathsf{end\text{-}device}$ party hides its "real" identity, whereas a server party does not. In Section 5, we present two protocols that allow an $\mathsf{end\text{-}device}$ party to behave either as $\mathsf{initiator}$ or $\mathsf{responder}$, and conversely a server can be either $\mathsf{responder}$ or $\mathsf{initiator}$. Therefore the notion of $\mathsf{end\text{-}device}$ and server is mainly used in the privacy experiment in order to indicate which party the adversary's goal is to find the identity.

[3] This is a technical feature of the SAKE and SAKE-AM protocols, which our PP-SAKE protocol is based on. In this regard, we refer the reader to [5, Section 6]. Note that alternatives exist, based on puncturable PRFs in order to update the master keys [15].

We put the following correctness requirements on the variables α, sk, sid and pid. For any two instances π_i^s, π_j^t, the following must hold:

$$\pi_i^s.\alpha = \texttt{accepted} \Rightarrow \pi_i^s.\mathsf{sk} \neq \perp \wedge \pi_i^s.\mathsf{sid} \neq \perp \tag{1}$$

$$\pi_i^s.\alpha = \pi_j^t.\alpha = \texttt{accepted} \wedge \pi_i^s.\mathsf{sid} = \pi_j^t.\mathsf{sid} \Rightarrow \begin{cases} \pi_i^s.\mathsf{sk} = \pi_j^t.\mathsf{sk} \\ \pi_i^s.\mathsf{pid} = P_j \\ \pi_j^t.\mathsf{pid} = P_i \end{cases} \tag{2}$$

Virtual identifier. In order to hide to the adversary which $\mathsf{end\text{-}device}$ party it is interacting with, the notion of virtual identifier is used. A virtual identifier $vid = P_i|P_j$ refers to two parties $P_i, P_j \in \mathcal{E}$, which are known to the adversary. The real involved party is designated by $\mathsf{realvid}(vid)$, depending on a secret bit $\mathsf{b} \in \{0,1\}$. This bit is sampled at initialisation of vid. If $vid.\mathsf{b} = 0$, then $\mathsf{realvid}(vid) = P_i$. If $vid.\mathsf{b} = 1$, then $\mathsf{realvid}(vid) = P_j$. In addition, $\mathsf{type}(vid) = \mathsf{end\text{-}device}$.

Adversarial queries. The adversary $\mathcal{A}$ is assumed to control the network, and interacts with the instances by issuing the following queries to them.

- $\mathsf{DrawParty}(P_i, P_j)$: this query creates a virtual identifier $vid = P_i|P_j$, adds vid to the list $\mathcal{L}_{\mathsf{vid}}$, and returns vid. If $P_i \notin \mathcal{E}$ or $P_j \notin \mathcal{E}$, then it returns $\perp$. If P_i or P_j are used in a virtual identifier already in $\mathcal{L}_{\mathsf{vid}}$, then it returns $\perp$.
- $\mathsf{NewSession}(id, \rho, id')$: this query creates a new instance π_i^s at party id, having role ρ, and intended partner id'. If $\mathsf{type}(id) = \mathsf{type}(id')$, the query aborts. If id is a virtual identifier, then the parent of π_i^s is $\mathsf{realvid}(id)$. If id' is a virtual identifier, then $\pi_i^s.\mathsf{pid} = \mathsf{realvid}(id')$. $\pi_i^s.\alpha$ is set to $\texttt{running}$. If $\rho = \mathsf{initiator}$, it produces the first message of the protocol which is returned to the adversary.
- $\mathsf{Send}(\pi_i^s, m)$: this query allows the adversary to send any message m to π_i^s. If $\pi_i^s.\alpha \neq \texttt{running}$, it returns $\perp$. Otherwise π_i^s responds according to the protocol specification.
- $\mathsf{Corrupt}(P_i)$: if $\mathsf{type}(P_i) = \mathsf{end\text{-}device}$, this query returns the long-term key $P_i.\mathsf{ltk}$ of P_i. If $\mathsf{type}(P_i) = \mathsf{server}$, this query returns all long-term keys $P_j.\mathsf{ltk}$, $P_j \in \mathcal{E}$, stored by P_i. If $\mathsf{Corrupt}(P_i)$ is the ν-th query issued by the adversary, then we say that P_i is *ν-corrupted*. For a party that has not been corrupted, we define $\nu = +\infty$. Moreover we say that a virtual identifier $vid = P_i|P_j$ is corrupted if either P_i or P_j is corrupted.
- $\mathsf{Reveal}(\pi_i^s)$: this query returns the session key $\pi_i^s.\mathsf{sk}$, and $\pi_i^s.\kappa$ is set to $\texttt{revealed}$.
- $\mathsf{Unmask}(\pi_i^s)$: this query returns the permanent identifier P_i of π_i^s's parent.
- $\mathsf{Test}(\pi_i^s)$: this query may be asked only once throughout the game. If $\pi_i^s.\alpha \neq \texttt{accepted}$, then it returns $\perp$. Otherwise it samples an independent key $sk_0 \xleftarrow{\$} \mathcal{K}$, and returns sk_b with $b = \pi_i^s.\mathsf{b}$, where $sk_1 = \pi_i^s.\mathsf{sk}$. The key sk_b is called the Test-*challenge-key*.
- $\mathsf{Free}(vid)$: this query removes vid from $\mathcal{L}_{\mathsf{vid}}$. Moreover, for any instance π_i^s such that either vid is the parent of π_i^s or $\pi_i^s.\mathsf{pid} = vid$, if $\pi_i^s.\alpha \in \{\perp, \texttt{running}\}$, then it sets $\pi_i^s.\alpha = \texttt{rejected}$.

The adversary is an active Man-in-the-Middle which can interact with parties, and adaptively issue queries. The adversary is granted the ability to query the used identities of arbitrary session partners (with the Unmask query). Our goal is to consider a strong adversary, in the sense that we allow as far as possible all queries (Corrupt, Reveal, Send, Unmask, etc.) except the queries that allow trivial attacks (i.e., attacks that allow the adversary to win, regardless of the design of the protocol).

Definition 1 (Partnership). *Two instances π_i^s and π_j^t are* partners *if $\pi_i^s.\mathsf{sid} = \pi_j^t.\mathsf{sid}$.*

A *privacy-preserving authenticated key exchange protocol* (PPAKE) is a two-party protocol satisfying the correctness requirements 1 and 2, and where the security is defined in terms of a PPAKE experiment played between a challenger and an adversary. This experiment uses the execution environment described above. The adversary can win the PPAKE experiment in one of three ways: (i) by making an instance accept maliciously, (ii) by guessing the secret bit of the Test-instance, or (iii) by guessing the secret bit of the privacy experiment.

Definition 2 (Entity Authentication (EA)). *An instance π_i^s of a protocol Π is said to have* accepted maliciously *in the PPAKE security experiment with intended partner P_j, if*

1. *$\pi_i^s.\alpha = \texttt{accepted}$ and $\pi_i^s.\mathsf{pid} = P_j$ when $\mathcal{A}$ issues its ν_0-th query,*
2. *P_i and P_j are ν- and ν'-corrupted with $\nu_0 < \nu$, $\nu_0 < \nu'$, and*
3. *there is no unique instance π_j^t such that π_i^s and π_j^t are partners.*

The adversary's advantage is defined as its winning probability:

$$\mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}} = \Pr[\mathcal{A} \textit{ wins the EA game}].$$

Definition 3 (Key Indistinguishability). *An adversary $\mathcal{A}$ against a protocol Π, that issues its* Test*-query to instance π_i^s during the PPAKE security experiment,* answers the Test-challenge-key correctly *if it terminates with output b', such that*

1. *$\pi_i^s.\alpha = \texttt{accepted}$ and $\pi_i^s.\mathsf{pid} = P_j$ when $\mathcal{A}$ issues its ν_0-th query,*
2. *$\pi_i^s.\kappa \neq \texttt{revealed}$ and P_i is ν-corrupted with $\nu_0 < \nu$,*
3. *for any partner instance π_j^t of π_i^s, we have that $\pi_j^t.\kappa \neq \texttt{revealed}$ and P_j is ν'-corrupted with $\nu_0 < \nu'$, and*
4. *$\pi_i^s.\mathsf{b} = b'$.*

The adversary's advantage is defined as

$$\mathsf{adv}_{\Pi}^{\mathsf{key\text{-}ind}} = \left| \Pr[\pi_i^s.\mathsf{b} = b'] - \frac{1}{2} \right|.$$

Note that the definition of key indistinguishability incorporates a requirement for forward secrecy.

Definition 4 (Privacy). *An adversary $\mathcal{A}$ against a protocol Π,* wins the privacy game *during the PPAKE security experiment, if it terminates with output (π_i^s, b'), such that*

- $\mathsf{type}(\pi_i^s) \neq \mathsf{type}(\pi_i^s.\mathsf{pid})$
- *If* $\mathsf{type}(\pi_i^s) = \mathsf{server}$*, let vid be the (virtual) identifier of* $\pi_i^s.\mathsf{pid}$*:*
  1. $\pi_i^s.\alpha = \texttt{accepted}$ *when $\mathcal{A}$ issues its ν_0-th query,*
  2. $\pi_i^u.\alpha = \texttt{accepted}$ *when $\mathcal{A}$ issues its ν_1-th query, where π_i^u is the instance created after π_i^s such that its parent and intended partner are the same as those of π_i^s,*
  3. *the parents of π_i^s and vid are ν- and ν'-corrupted with $\nu_1 < \nu$, $\nu_0 < \nu'$,*
  4. *$\mathcal{A}$ did not issue an* Unmask *query to π_j^t for any instance π_j^t such that π_j^t is partnered with π_i^s and $\mathsf{type}(\pi_j^t) = \mathsf{end\text{-}device}$, and*
  5. *vid*$.\mathsf{b} = b'$.
- *If* $\mathsf{type}(\pi_i^s) = \mathsf{end\text{-}device}$*, let vid be the (virtual) identifier of the parent of π_i^s:*
  1. $\pi_i^s.\alpha = \texttt{accepted}$ *when $\mathcal{A}$ issues its ν_0-th query,*
  2. $\pi_j^v.\alpha = \texttt{accepted}$ *when $\mathcal{A}$ issues its ν_1-th query, where π_j^v is created after π_j^t for any partner π_j^t of π_i^s, such that $\pi_j^v.\mathsf{pid}$ is π_i^s's parent and π_j^v's parent is $\pi_i^s.\mathsf{pid}$,*
  3. *vid and $\pi_i^s.\mathsf{pid}$ are ν- and ν'-corrupted with $\nu_0 < \nu$, $\nu_1 < \nu'$,*
  4. *$\mathcal{A}$ did not issue an* Unmask *query to π_i^s, and*
  5. *vid*$.\mathsf{b} = b'$.

The adversary's advantage is defined as

$$\mathsf{adv}_{\Pi}^{\mathsf{privacy}} = \left| \Pr[vid.\mathsf{b} = b'] - \frac{1}{2} \right|.$$

Definitions 2, 3, and 4 allow the adversary to corrupt an instance involved in the security experiment (after some time, in order to exclude trivial attacks). Therefore, protocols secure with respect to Definition 5 below provide *forward secrecy*. We do not allow the targeted instance to be corrupted before it accepts. That is, this security model does not capture key-compromise impersonation attacks (KCI) [12] since that would allow trivially breaking key exchange protocols solely based on shared symmetric keys.

Definition 5 (PPAKE Security). *We say that a two-party protocol Π is a secure PPAKE protocol if Π satisfies the correctness requirements 1 and 2, and for all probabilistic polynomial time adversary $\mathcal{A}$,* $\mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}$*,* $\mathsf{adv}_{\Pi}^{\mathsf{key\text{-}ind}}$*, and* $\mathsf{adv}_{\Pi}^{\mathsf{privacy}}$ *are a negligible function of the security parameter.*

4.2 Security Definitions of the Building Blocks

In our proofs, we rely upon standard security definitions. The security definition of a pseudo-random function (PRF) is taken from Bellare, Desai, Jokipii, and Rogaway [8], and that of a MAC strongly unforgeable under chosen-message attacks from Bellare and Namprempre [9]. We rely also on the definition of matching conversations initially proposed by Bellare and Rogaway [10], and modified by Jager, Kohlar, Schäge, and Schwenk [26].

5 Privacy-Preserving SAKE/SAKE-AM

In this section we present the protocol obtained when applying to [1] the mitigations described in Section 3. We call these corrected versions respectively Privacy-preserving SAKE (PPSAKE) and Privacy-preserving SAKE-AM (PPSAKE-AM).

Description. The protocol PPSAKE is depicted by Figure 4. It illustrates the generic case when party A is a server communicating with a set of end-devices (parties B). As in [1], B uses either an ephemeral identity parameter id_B (which evolves the same way as its master keys K and K') or a pseudo-random value, depending if the previous protocol run has completed successfully (this is tracked with the flag ϕ, initialised to 0). $\bar{id}_B = id_B$ allows A to retrieve the set of parameters corresponding to B in its database db. For each party B, A stores the identity parameter of three consecutive epochs, as it is done with the authentication master key (each *entry* in db is of the form: $K, (id_{B,j}, K'_j), (id_{B,j-1}, K'_{j-1}), (id_{B,j+1}, K'_{j+1})$).

When B transmits a pseudo-random value $\bar{id}_B$ (instead of id_B), A must explore (in constant time) the database (function find-entry) in order to find the matching authentication master key (i.e., which allows verifying the MAC tag τ_B). This happens only if the previous session (between A and B) was not successful.

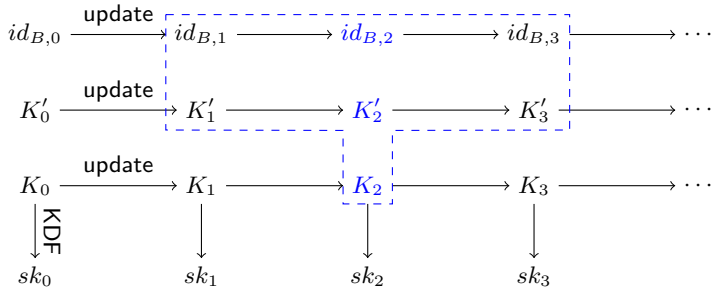

Fig. 3: Party A stores authentication master keys and identifiers corresponding to three consecutive epochs ($j-1$, j, $j+1$), and one derivation master key (illustration with $j = 2$ with the blue dashed box). Party B stores one sample of each master key, and identifier (in blue).

The same value $\bar{id}_B$ is used in a given session, and a new value is used in the next session (see Figure 3). The identity parameter $\bar{id}_B$ appears in all messages (but the first one in PPSAKE). The purpose is to allow A to recognise which keys to use (when $\bar{id}_B$ is output by the update function), but also to allow B detecting which messages sent by A are intended to it. Likewise, with $\bar{id}_B$ (equal

A $(id_A,\ \mathsf{db})$		B $(id_B,\ K,\ K',\ \phi)$
$r_A \xleftarrow{\$} \{0,1\}^\lambda$		
	$\xrightarrow{id_A \| r_A}$	

		`if` $(\phi = 0)$
		  $i\bar{d}_B \leftarrow id_B$
		  $\phi \leftarrow 1$
		`else if` $(\phi = 1)$
		  $i\bar{d}_B \xleftarrow{\$} \{0,1\}^\lambda$
		
		$r_B \xleftarrow{\$} \{0,1\}^\lambda$
		$\tau_B \leftarrow \mathsf{Mac}(K', i\bar{d}_B \| id_A \| r_B \| r_A)$
		$m_B \leftarrow i\bar{d}_B \| r_B \| \tau_B$

	$\xleftarrow{m_B}$	

`if` $(i\bar{d}_B \in \mathsf{db.id})$
  $entry \leftarrow$ get corresponding entry
  `if` $(\mathsf{verif\text{-}entry}(entry, m_B) = \mathtt{false})$
  abort
`else`
  $entry \leftarrow \mathsf{find\text{-}entry}(m_B)$
  `if` $(entry = \emptyset)$
  abort		

$\tau_A \leftarrow \mathsf{Mac}(K', \epsilon \| id_A \| i\bar{d}_B \| r_A \| r_B)$ $m_A \leftarrow \epsilon \| i\bar{d}_B \| \tau_A$		
	$\xrightarrow{m_A}$	

		`if` $(\mathsf{Vrf}(K', \epsilon \| id_A \| i\bar{d}_B \| r_A \| r_B, \tau_A) = \mathtt{false})$
		  abort
		`if` $(\epsilon = 1)$
		  upd_B
		
		$\mathsf{kdf};\ \mathsf{upd}_B$
		$\phi \leftarrow 0$
		$\tau'_B \leftarrow \mathsf{Mac}(K', i\bar{d}_B \| r_B \| r_A)$
		$m'_B \leftarrow i\bar{d}_B \| \tau'_B$

	$\xleftarrow{m'_B}$	

`if` $(\epsilon = 0)$
  $K' \leftarrow K'_j$
  `if` $(\mathsf{Vrf}(K', i\bar{d}_B \| r_B \| r_A, \tau'_B) = \mathtt{false})$
  abort
`else if` $(\epsilon = 1)$
  $K' \leftarrow K'_{j+1}$
  `if` $(\mathsf{Vrf}(K', i\bar{d}_B \| r_B \| r_A, \tau'_B) = \mathtt{false})$
  abort
  $\mathsf{kdf};\ \mathsf{upd}_A$		

$\tau'_A \leftarrow \mathsf{Mac}(K', i\bar{d}_B \| r_A \| r_B)$ $m'_A \leftarrow i\bar{d}_B \| \tau'_A$		
	$\xrightarrow{m'_A}$	
		`if` $(\mathsf{Vrf}(K', i\bar{d}_B \| r_A \| r_B, \tau'_A) = \mathtt{false})$ abort

Fig. 4: The Privacy-preserving SAKE protocol

A $(id_A,$ db, htable)		B $(id_B,\ K,\ K',\ \phi)$
		`if` $(\phi = 0)$ $\quad i\bar{d}_B \leftarrow id_B$ $\quad \phi \leftarrow 1$ `else if` $(\phi = 1)$ $\quad i\bar{d}_B \xleftarrow{\$} \{0,1\}^\lambda$ $r_B \xleftarrow{\$} \{0,1\}^\lambda$ $\tau_B \leftarrow \mathsf{Mac}(K', i\bar{d}_B \| id_A \| r_B)$ $m_B \leftarrow i\bar{d}_B \| r_B \| \tau_B$
	$\xleftarrow{m_B}$	
`if` $(i\bar{d}_B \in$ db.id) $\quad entry \leftarrow$ get corresponding entry $\quad$ `if` (verif-table(htable, m_B) = `true`) $\quad\quad$ abort $\quad$ `if` (verif-entry($entry, m_B$) = `false`) $\quad\quad$ abort `else` $\quad entry \leftarrow$ find-entry(m_B) $\quad$ `if` $(entry = \emptyset)$ $\quad\quad$ abort $\quad$ `if` (verif-table(htable, m_B) = `true`) $\quad\quad$ abort insert-table(htable, m_B) $r_A \xleftarrow{\$} \{0,1\}^\lambda$ $\tau_A \leftarrow \mathsf{Mac}(K', \epsilon \| id_A \| i\bar{d}_B \| r_A \| r_B)$ $m_A \leftarrow \epsilon \| i\bar{d}_B \| r_A \| \tau_A$		
	$\xrightarrow{m_A}$	
		`if` $(\mathsf{Vrf}(K', \epsilon \| id_A \| i\bar{d}_B \| r_A \| r_B, \tau_A) =$ `false`$)$ $\quad$ abort `if` $(\epsilon = 1)$ $\quad \mathsf{upd}_B$ kdf; upd_B $\phi \leftarrow 0$ $\tau'_B \leftarrow \mathsf{Mac}(K', i\bar{d}_B \| id_A \| r_B \| r_A)$ $m'_B \leftarrow i\bar{d}_B \| \tau'_B$
	$\xleftarrow{m'_B}$	
`if` $(\epsilon = 0)$ $\quad K' \leftarrow K'_j$ $\quad$ `if` $(\mathsf{Vrf}(K', i\bar{d}_B \| id_A \| r_B \| r_A, \tau'_B) =$ `false`$)$ $\quad\quad$ abort `else if` $(\epsilon = 1)$ $\quad K' \leftarrow K'_{j+1}$ $\quad$ `if` $(\mathsf{Vrf}(K', i\bar{d}_B \| id_A \| r_B \| r_A, \tau'_B) =$ `false`$)$ $\quad\quad$ abort $\quad$ kdf; upd_A $\tau'_A \leftarrow \mathsf{Mac}(K', i\bar{d}_B \| r_A \| r_B)$ $m'_A \leftarrow i\bar{d}_B \| \tau'_A$		
	$\xrightarrow{m'_A}$	
		`if` $(\mathsf{Vrf}(K', i\bar{d}_B \| r_A \| r_B, \tau'_A) =$ `false`$)$ $\quad$ abort

Fig. 5: The Privacy-preserving SAKE-AM protocol

to id_B or pseudo-random), A can correlate the messages m_B, m'_B and the corresponding parameters in database. The identity parameter id_A used by A is explicit, and never changed as the privacy property aims at protecting B.

In PPSAKE-AM (see Figure 5), party A stores efficiently (e.g., using a Bloom or a Cuckoo filter) the messages m_B it receives (this is not necessary in PPSAKE). Upon reception of a message m_B, A verifies if it has already been received (these operations are done respectively with functions insert-table and verif-table). This prevents A from responding to a replayed message m_B. We use a global table. Note that using one table per entry in the database db (i.e., one table per end-device party) may lead to privacy breaches depending on how the privacy experiment is defined (in particular if the table can be revealed).

We observe that, depending on the communication layer, it may not be necessary to include $\bar{id}_B$ in all messages. For instance, if the messages are sent through a radio link, A and B can negotiate a specific radio frequency they will use during the session. Likewise, if the messages are sent through a wired link, some ephemeral per-session identifier (e.g., IP address or any equivalent) can be used by A and B to discriminate the messages intended to them. Nonetheless, in order to make the protocol agnostic with respect to the communication layer, we opt for an ephemeral identifier $\bar{id}_B$ in the messages.

As in [5], using PPSAKE-AM together with PPSAKE (both protocols are based on the same inner functions) allows any party to be either initiator or responder of a session, such that the smallest amount of calculation is always done by the same party (i.e., the low-resource end-device).

Notations. The notations upd_A, and upd_B are defined as follows:

- upd_A corresponds to
  1. $K \leftarrow \mathsf{update}(K)$
  2. $id_{B,j-1} \leftarrow id_{B,j}$
  3. $id_{B,j} \leftarrow id_{B,j+1}$
  4. $id_{B,j+1} \leftarrow \mathsf{update}(K'_{j+1}, id_{B,j+1})$
  5. $K'_{j-1} \leftarrow K'_j$
  6. $K'_j \leftarrow K'_{j+1}$
  7. $K'_{j+1} \leftarrow \mathsf{update}(K'_{j+1})$
- upd_B corresponds to
  1. $K \leftarrow \mathsf{update}(K)$
  2. $id_B \leftarrow \mathsf{update}(K', id_B)$
  3. $K' \leftarrow \mathsf{update}(K')$

The function verif-entry (see Figure 6) takes as input an entry $entry \in \mathsf{db}$, and a message m_B (we assume that the other values used in verif-entry are "global" parameters). It outputs `true` if $entry$ allows verifying correctly m_B.

The function find-entry (see Figure 7) takes as input a message $m_B = \bar{id}_B \| r_B \| \tau_B$, and outputs either an entry $entry \in \mathsf{db}$ or $\emptyset$.

The function verif-table takes as input a hash table htable, and a value x, and outputs `true` if x is present in htable. Otherwise it returns `false`.

The function insert-table takes as input a hash table htable, and a value x, and inserts x into htable.

```
if (Vrf(K'_j, id̄_B ‖ id_A ‖ r_B ‖ r̃_A, τ_B) = true)
    δ_AB ← 0
    K' ← K'_j
    kdf
    upd_A
    ϵ ← 0
    return true
else if (Vrf(K'_{j−1}, id̄_B ‖ id_A ‖ r_B ‖ r̃_A, τ_B) = true)
    δ_AB ← 1
    K' ← K'_{j−1}
    ϵ ← 1
    return true
else if (Vrf(K'_{j+1}, id̄_B ‖ id_A ‖ r_B ‖ r̃_A, τ_B) = true)
    δ_AB ← −1
    K' ← K'_{j+1}
    upd_A
    kdf
    upd_A
    ϵ ← 0
    return true
else
    return false
```

Fig. 6: Pseudo-code of function verif-entry. Elements surrounded by a blue dashed box appear only in PPSAKE (not in PPSAKE-AM).

```
foreach entry ∈ db
    if (verif-entry(entry, m_B) = true)
        return entry
return ∅
```

Fig. 7: Pseudo-code of function find-entry

6 Security of Privacy-Preserving SAKE/SAKE-AM

In this section we prove that PPSAKE and PPSAKE-AM are secure PPAKE protocols according to Definition 5. We refer the reader to [5] regarding the *soundness* of PPSAKE and PPSAKE-AM (the proof is the same as that of SAKE and SAKE-AM in this regard).

In order to prove that the protocol PPSAKE/PPSAKE-AM is a secure PPAKE protocol, we use the execution environment described in Section 4.1. We define the partnering between two instances with the notion of *matching conversations*. That is, we define sid to be the transcript, in chronological order, of all the (valid) messages sent and received by an instance during the key exchange, but, possibly, the last one. Furthermore, we choose the function update to be a keyed PRF, that is $\mathsf{update} : (k, x) \mapsto \mathsf{PRF}(k, x)$, and we define:

- $\mathsf{update}(k) = \mathsf{PRF}(k, c)$ if k is a derivation (K) or authentication (K') master key, and c is some (constant) value
- $\mathsf{update}(K', id_B) = \mathsf{PRF}(K', id_B)$

Theorem 1. *The protocol $\Pi \in \{PPSAKE, PPSAKE\text{-}AM\}$ is a secure PPAKE protocol, and for any probabilistic polynomial time adversary $\mathcal{A}$ in the PPAKE security experiment against Π*

$$\mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}} \leq nq\left((nq-1)2^{-\lambda} + (n_{\mathsf{E}}(q-1)+2)\mathsf{adv}_{\mathsf{update}}^{\mathsf{prf}} + (n_{\mathsf{E}}+1)\mathsf{adv}_{\mathsf{Mac}}^{\mathsf{suf\text{-}cma}}\right)$$

$$\mathsf{adv}_{\Pi}^{\mathsf{key\text{-}ind}} \leq nq\left((q-1)\mathsf{adv}_{\mathsf{update}}^{\mathsf{prf}} + \mathsf{adv}_{\mathsf{KDF}}^{\mathsf{prf}}\right) + \mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}$$

$$\mathsf{adv}_{\Pi}^{\mathsf{privacy}} \leq nq\left(q \cdot \mathsf{adv}_{\mathsf{update}}^{\mathsf{prf}} + 2\mathsf{adv}_{\mathsf{Mac}}^{\mathsf{prf}} + 2^{-\kappa}\right) + \mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}$$

where n_{E} is the number of end-device *parties, n_{S} the number of* server *parties, $n = n_{\mathsf{E}} + n_{\mathsf{S}}$, q the maximum number of instances (sessions) per party, κ the size of the derivation master key K, λ the size of the pseudo-random values (r_A, r_B), and $\mathsf{adv}_{\mathsf{update}}^{\mathsf{prf}}$, $\mathsf{adv}_{\mathsf{Mac}}^{\mathsf{suf\text{-}cma}}$, $\mathsf{adv}_{\mathsf{KDF}}^{\mathsf{prf}}$, and $\mathsf{adv}_{\mathsf{Mac}}^{\mathsf{prf}}$ are the advantage of an adversary to break respectively the PRF-security of* update*, the SUF-CMA-security of* Mac*, the PRF-security of* KDF*, and the PRF-security of* Mac*.*

A sketch of proof of Theorem 1 is given below. The full proof is presented in Section A.

Proof (Sketch). We proceed through a sequence of games between a challenger and an adversary $\mathcal{A}$. First we consider the entity authentication experiment. We use the following hops.

Game 0 corresponds to the entity authentication security experiment described in Section 4.1.

In *Game 1*, the challenger aborts if there exists an instance that chooses a random value r_A or r_B that is not unique. There is at most $n \times q$ random values, each uniformly drawn at random in $\{0,1\}^{\lambda}$. Therefore, the two games are equivalent up to a collision term $\frac{nq(nq-1)}{2^{\lambda}}$.

In *Game 2*, the challenger tries to guess which instance π will be the first to accept maliciously, and aborts the game if the guess is wrong. The security loss is a factor nq.

In *Game 3*, the challenger aborts if the initiator (resp. responder) instance π ever receives a message m_B (resp. m_A), but no instance having a matching conversation to π has output that message. Here, we reduce the probability of this event to the security of the functions Mac (used to compute the MAC tags) and update (used to update the MAC key). Since there are at most q sessions per party, and accounting that the initiator may try all authentication master keys present in its database (i.e., n_{E} keys), the security loss is equal to $n_{\mathsf{E}}\left((q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}} + \mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Mac}}\right)$.

In *Game 4*, the challenger aborts if the targeted instance π ever receives a valid message τ'_B (resp. τ'_A), but no instance having a matching conversation to π has output that message. We reduce the probability of the adversary to win this game to the security of the Mac function used to compute the message τ'_B (resp. τ'_A). In turn we rely on the randomness of the MAC key, hence on the security of the function update used to update the MAC key K'. Hence a loss equal to $2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}} + \mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Mac}}$.

To that point, the only way for the adversary to make π accept maliciously is to send a valid message τ'_B (resp. τ'_A) different from all the messages sent by all the instances. However, in such a case, the challenger aborts. Hence the adversary has no chance to win.

Now we prove the key indistinguishability security.

Game 0 corresponds to the key indistinguishability experiment described in Section 4.1.

In *Game 1*, the challenger aborts the experiment and chooses $b' \in \{0,1\}$ uniformly at random if there exists an instance that accepts maliciously. In other words, in this game we make the same modifications as in the games performed during the entity authentication proof. Hence a loss equal to $\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{PPSAKE}$.

In *Game 2*, the challenger tries to guess which instance is targeted by the adversary, and aborts the game if the guess is wrong. The security loss is a factor nq.

In *Game 3*, we reduce the advantage of the adversary to win this game to the security of the function KDF used to compute the session key. In turn we rely on the PRF-security of the function update used to update the derivation master key K. Since there is at most q sessions per party, this implies a security loss equal to $(q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}} + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}}$.

To that point, the adversary can do no better than guessing.

Now we prove the privacy security.

Game 0 corresponds to the privacy experiment described in Section 4.1.

In *Game 1*, the challenger aborts the experiment and chooses $b' \in \{0,1\}$ uniformly at random if there exists an instance that accepts maliciously. Hence a loss $\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{PPSAKE}$.

In *Game 2*, the challenger tries to guess which instance is targeted by the adversary, and aborts the game if the guess is wrong. The loss is a factor nq.

In *Game 3*, the challenger aborts if the adversary succeeds in guessing the derivation master key (with this key and a $\mathsf{Corrupt}$ query, the adversary can recognise the "real" party involved in the session). Hence a loss equal to $\frac{1}{2^\kappa}$.

In *Game 4*, the challenger aborts if the adversary succeeds in correlating any value (besides id_A) exchanged during the current session with any value exchanged during a previous session. In order to claim that the values exchanged during the session (in particular id_B and the MAC tags) are random and independent of all previous sessions, we rely upon the PRF-security of update and Mac used to compute respectively id_B and the MAC tags. These function are keyed with (different values of) the authentication master key. In turn the latter is computed with update. Overall this implies a security loss equal to $q \cdot \mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}} + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{Mac}}$

To that point the adversary can do no better than guessing the secret bit of the privacy experiment. □

7 Conclusion

In this paper we have investigated the field of privacy-preserving authenticated key exchange protocols (PPAKE).

First we have made a cryptographic analysis of a previous PPAKE protocol intended to the IoT [1], and shown that most of its security properties, including privacy, are broken. Furthermore we have described countermeasures that allow preventing these vulnerabilities. The attacks that we exhibit question the correctness of the security proofs provided in [1], and highlight the importance of using a suitable security model.

Secondly, we have presented a security model which captures the security properties of a PPAKE protocol: entity authentication, key indistinguishability, forward secrecy, and privacy. The approach that we take guarantees that the different security properties are independent of each other, which yields a strong security model. We do think that this security model can serve as a tool to analyse other authenticated key exchange protocols that implement mechanisms to guarantee privacy.

Finally, we have described a PPAKE protocol in the symmetric-key setting which is suitable for constrained devices. We have formally proved the security of this protocol in our strong model.

References

1. Aghili, S.F., Jolfaei, A.A., Abidin, A.: SAKE$^+$: Strengthened symmetric-key authenticated key exchange with perfect forward secrecy for IoT. Cryptology ePrint Archive, Report 2020/778, 20200714:112142 (2020)
2. ANSSI: Should Quantum Key Distribution be Used for Secure Communications? (May 2020)
3. Arfaoui, G., Bultel, X., Fouque, P.A., Nedelcu, A., Onete, C.: The privacy of the TLS 1.3 protocol. PoPETs **2019**(4), 190–210 (Oct 2019)

4. Ashur, T., Delvaux, J., Lee, S., Maene, P., Marin, E., Nikova, S., Reparaz, O., Rozic, V., Singelée, D., Yang, B., Preneel, B.: A privacy-preserving device tracking system using a low-power wide-area network. In: Capkun, S., Chow, S.S.M. (eds.) CANS 17. LNCS, vol. 11261, pp. 347–369. Springer, Heidelberg (Nov / Dec 2017)
5. Avoine, G., Canard, S., Ferreira, L.: Symmetric-key authenticated key exchange (SAKE) with perfect forward secrecy. In: Jarecki, S. (ed.) CT-RSA 2020. LNCS, vol. 12006, pp. 199–224. Springer, Heidelberg (Feb 2020)
6. Avoine, G., Coisel, I., Martin, T.: Time measurement threatens privacy-friendly RFID authentication protocols. In: Yalcin, S.B.O. (ed.) Radio Frequency Identification: Security and Privacy Issues - 6th International Workshop, RFIDSec 2010. LNCS, vol. 6370, pp. 138–157. Springer (2010)
7. Avoine, G., Coisel, I., Martin, T.: Untraceability Model for RFID. IEEE Transactions on Mobile Computing **13**(10), 9 (Oct 2014)
8. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: 38th FOCS. pp. 394–403. IEEE Computer Society Press (Oct 1997)
9. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. Journal of Cryptology **21**(4), 469–491 (Oct 2008)
10. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (Aug 1994)
11. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (May / Jun 2006)
12. Blake-Wilson, S., Johnson, D., Menezes, A.: Key agreement protocols and their security analysis. In: Darnell, M. (ed.) 6th IMA International Conference on Cryptography and Coding. LNCS, vol. 1355, pp. 30–45. Springer, Heidelberg (Dec 1997)
13. Blanchet, B., Smyth, B., Cheval, V., Sylvestre, M.: ProVerif 2.01: Automatic cryptographic protocol verifier, user manual and tutorial (April 2020)
14. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13**(7), 422–426 (July 1970)
15. Boyd, C., Davies, G.T., de Kock, B., Gellert, K., Jager, T., Millerjord, L.: Symmetric key exchange with full forward security and robust synchronization. Cryptology ePrint Archive, Report 2021/702 (2021)
16. Brzuska, C., Jacobsen, H., Stebila, D.: Safely exporting keys from secure channels: On the security of EAP-TLS and TLS key exporters. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 670–698. Springer, Heidelberg (May 2016)
17. Canard, Sébastien Coisel, I.: Data Synchronization in Privacy-Preserving RFID Authentication Schemes. In: Radio Frequency Identification: Security and Privacy Issues - 4th International Workshop, RFIDSec 2008 (2008)
18. Canard, S., Coisel, I., Etrog, J., Girault, M.: Privacy-preserving RFID systems: Model and constructions. Cryptology ePrint Archive, Report 2010/405 (2010)
19. Dimitriou, T.: Key Evolving RFID Systems. Ad Hoc Netw. **37**(P2), 195–208 (February 2016)
20. Fan, B., Andersen, D.G., Kaminsky, M., Mitzenmacher, M.: Cuckoo filter: Practically better than bloom. In: Seneviratne, A., Diot, C., Kurose, J., Chaintreau, A., Rizzo, L. (eds.) Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, CoNEXT 2014. pp. 75–88. ACM (2014)

21. Fischlin, M., Günther, F.: Replay attacks on zero round-trip time: The case of the TLS 1.3 handshake candidates. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 60–75. IEEE (April 2017)
22. Fouque, P.A., Onete, C., Richard, B.: Achieving better privacy for the 3GPP AKA protocol. PoPETs **2016**(4), 255–275 (Oct 2016)
23. Hedbom, H.: A Survey on Transparency Tools for Enhancing Privacy. In: Matyáš, V., Fischer-Hübner, S., Cvrček, D., Švenda, P. (eds.) The Future of Identity in the Information Society. pp. 67–82. Springer (2009)
24. Hermans, J., Pashalidis, A., Vercauteren, F., Preneel, B.: A new RFID privacy model. In: Atluri, V., Díaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 568–587. Springer, Heidelberg (2011)
25. Huang, H.F., Yu, P.K., Liu, K.C.: A privacy and authentication protocol for mobile RFID system. In: International Symposium on Independent Computing – ISIC 2014 (2014)
26. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 273–293. Springer, Heidelberg (Aug 2012)
27. Juels, A.: RFID Security and Privacy: A Research Survey. IEEE J.Sel. A. Commun. **24**(2), 381–394 (September 2006)
28. Juels, A., Weis, S.A.: Defining Strong Privacy for RFID. In: Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07). pp. 342–347 (2007)
29. Malina, L., Srivastava, G., Dzurenda, P., Hajny, J., Ricci, S.: A Privacy-Enhancing Framework for Internet of Things Services. In: Liu, J.K., Huang, X. (eds.) Network and System Security. pp. 77–97. Springer International Publishing (2019)
30. Ouafi, K., Phan, R.C.W.: Traceable privacy of recent provably-secure RFID protocols. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 08. LNCS, vol. 5037, pp. 479–489. Springer, Heidelberg (Jun 2008)
31. Ray, A.K., Bagwari, A.: Study of smart home communication protocol's and security privacy aspects. In: 7th International Conference on Communication Systems and Network Technologies (CSNT). pp. 240–245 (2017)
32. Rescorla, E.: The transport layer security (TLS) protocol version 1.3 (August 2018)
33. Schäge, S., Schwenk, J., Lauer, S.: Privacy-preserving authenticated key exchange and the case of IKEv2. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 567–596. Springer, Heidelberg (May 2020)
34. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004)
35. Song, T., Li, R., Mei, B., Yu, J., Xing, X., Cheng, X.: A Privacy Preserving Communication Protocol for IoT Applications in Smart Homes. IEEE Internet of Things Journal **4**(6), 1844–1852 (2017)
36. Vaudenay, S.: On privacy models for RFID. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 68–87. Springer, Heidelberg (Dec 2007)
37. You, I., Kwon, S., Choudhary, G., Sharma, V., Seo, J.T.: An Enhanced LoRaWAN Security Protocol for Privacy Preservation in IoT with a Case Study on a Smart Factory-Enabled Parking System. Sensors **18**(6) (2018)
38. Ziegeldorf, J.H., Morchon, O.G., Wehrle, K.: Privacy in the Internet of Things: threats and challenges. Secur. Commun. Networks **7**(12), 2728–2742 (2014)

A Extended Proof for PPSAKE/PPSAKE-AM

We give a proof of Theorem 1 for Π = PPSAKE. The reasoning for Π = PPSAKE-AM is similar.

A.1 Extended Proof for Entity Authentication

In order for an initiator instance π_i^s at some party P_i to accept, two valid messages (i.e., with valid MAC tags) must be received by π_i^s (m_B and τ'_B). We reduce the security of the Mac function to the (in)ability to forge a valid output. Therefore we use the fact that the key K' is random. By assumption, the genuine value of K' (i.e., the value used during the first session between two same parties) is uniformly chosen at random. Yet K' (and K) is updated throughout the session with the function update. If K' is random, we can rely on the pseudo-randomness of $\mathsf{update}(\cdot) = \mathsf{PRF}(\cdot,\cdot)$. In turn, since $\mathsf{PRF}(K',\cdot)$ can be replaced with a truly random function, its output (updated K') is random. Therefore, one can rely upon the pseudo-randomness of the function update keyed with this new value K', and so forth. Each transition (i.e., each update of K') implies a loss equal to $\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}$ corresponding to the ability of an adversary $\mathcal{A}_0$ to distinguish update from a random function.

If P_i is synchronised with the responder ($\delta_{AB} = 0$), P_i updates its master keys once (upon reception of m_B). If P_i is in advance ($\delta_{AB} = 1$), it updates its keys at most once (if a valid message τ'_B is received). If P_i is late ($\delta_{AB} = -1$), it updates its keys twice. Yet, in that case, P_i did not update its keys during the previous session. Therefore, on average, P_i updates its keys at most once per session. Hence, when the u-th session starts, P_i has updated its keys at most $u-1$ times on average, and, upon reception of τ'_B, P_i updates the keys at most two times.

This is similar regarding the responder. A responder instance π_j^t at some party P_j accepts only if the two messages m_A and τ'_A are valid. Upon reception of a valid message m_A, the keys are updated once ($\epsilon = 0$) or twice ($\epsilon = 1$). In the latter case, the keys have not been updated during the previous session. This means that the keys are updated on average at most once per session. Therefore, when the u-th session starts, P_j has updated its keys at most $u-1$ times on average, and, upon reception of m_A, the keys are updated at most two times.

We can now proceed with the proof. We proceed through a sequence of games [11, 34], where each consecutive game aims at reducing the challenger's dependency on the functions Mac, update and KDF. We first prove the entity authentication security. Let E_i be the event that the adversary win the entity authentication experiment, as defined by Definition 2, in Game i.

Game 0. This game corresponds to the entity authentication security experiment described in Section 4.1. Therefore

$$\Pr[E_0] = \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{PPSAKE}$$

Game 1. The challenger aborts if there exists any instance that chooses a random value r_A or r_B that is not unique. There is at most $n \times q$ random values (with $n = n_{\mathsf{E}} + n_{\mathsf{S}}$), each uniformly drawn at random in $\{0,1\}^\lambda$. Therefore the probability that at least two random values be equal is at most $\frac{nq(nq-1)}{2^\lambda}$. Hence

$$\Pr[E_0] \leq \Pr[E_1] + \frac{nq(nq-1)}{2^\lambda}$$

Game 2. The challenger tries to guess which instance will be the first to accept maliciously. If the guess is wrong, the game is aborted. The number of instances is at most nq. Therefore

$$\Pr[E_2] = \Pr[E_1] \times \frac{1}{nq}$$

Game 3. Let π be the instance targeted by the adversary. In this game, we add an abort rule. The challenger aborts the experiment if π ever receives a valid message m_B (resp. m_A) if it is an initiator (resp. responder) instance, but no instance having a matching conversation to π has output that message. We reduce the probability of this event to the security of the functions Mac and update. As explained above, when the u-th session starts, the master keys have been updated at most $u-1$ times already. The genuine value of K' is uniformly chosen at random. In order to be able to replace, during the current session, the key used to compute the MAC tag in m_A (resp. m_B) with a random value, one must rely upon the pseudo-randomness of the function update that outputs (the new value of) K'. In turn, this relies upon the (previous) key K' being random (and on the pseudo-randomness of update). Therefore, in order to replace K' with a random value one must take into account the successive losses $\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}$, each corresponding to the ability of an adversary $\mathcal{A}_0$ to distinguish the function update (keyed with a different key K') from a random function. Since there is at most q sessions, this loss is at most $(q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}$. Then we reduce the probability of the adversary $\mathcal{A}$ to win this game to the ability of an adversary $\mathcal{A}_1$ to forge a valid tag τ_B (resp. τ_A).

Therefore, we replace each function $\mathsf{update}(K') = \mathsf{PRF}(K', c)$ (keyed with a different key K' throughout the, at most, $q-1$ successive sessions established, prior to that current session, by the same party that owns π) with truly random functions $\mathsf{F}_0^{\mathsf{update}}$, ..., $\mathsf{F}_{q-2}^{\mathsf{update}}$. Moreover, if an instance uses the same key $K' = K'_i$, $0 \leq i < q-1$, to key update, then we replace update with the corresponding random function $\mathsf{F}_i^{\mathsf{update}}$. Since, to that point, the key $K' = K'_{q-1}$ used to compute the authentication tag τ_B (resp. τ_A) is random, we reduce the ability of $\mathcal{A}$ to win to the security of the Mac function.

This reasoning holds if the initiator verifies the MAC tag with one authentication master key. However, it may happen that the initiator tries multiple authentication keys (when $\phi = 1$), possibly all of them, in order to find the proper one. Accounting that the initiator stores at most n_{E} entries (hence n_{E} authentication master keys) in its database, we have that

$$\Pr[E_2] \leq \Pr[E_3] + n_{\mathsf{E}} \times \left((q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}} + \mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Mac}} \right)$$

Game 4. The challenger aborts the experiment if π ever receives a valid message τ'_B (resp. τ'_A), but no instance having a matching conversation to π has output that message. Between the message m_B (resp. m_A) being received by π, and the message τ'_B (resp. τ'_A) being received by π, the master keys are updated at most twice. We reduce the probability of the adversary to win this game to the security of the Mac function used to compute the message τ'_B (resp. τ'_A). In turn we must rely on the randomness of the MAC key, hence on the security of the function update used to update the MAC key K' (recall that, due to Game 3, the current key K' is random). Therefore

$$\Pr[E_3] \leq \Pr[E_4] + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}} + \mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Mac}}$$

To that point, the only way for the adversary to make π accept maliciously is to send a valid message τ'_B (resp. τ'_A) different from all the messages sent by all the instances. However, in such a case, the challenger aborts. Therefore $\Pr[E_4] = 0$.

Collecting all the probabilities from Game 0 to 4, we have that

$$\begin{aligned}
\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{PPSAKE} &= \Pr[E_0] \\
&\leq \Pr[E_1] + \frac{nq(nq-1)}{2^\lambda} \\
&= nq\Pr[E_2] + \frac{nq(nq-1)}{2^\lambda} \\
&\leq nq\left(\Pr[E_3] + n_{\mathsf{E}}\left((q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}} + \mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Mac}}\right)\right) + \frac{nq(nq-1)}{2^\lambda} \\
&\leq nq\left(\Pr[E_4] + (n_{\mathsf{E}}(q-1)+2)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}} + (n_{\mathsf{E}}+1)\mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Mac}}\right) \\
&\qquad + \frac{nq(nq-1)}{2^\lambda} \\
&= nq\left((nq-1)2^{-\lambda} + (n_{\mathsf{E}}(q-1)+2)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}} + (n_{\mathsf{E}}+1)\mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Mac}}\right)
\end{aligned}$$

A.2 Extended Proof for Key Indistinguisability

Let E_i be the event that an adversary win the key indistinguishability experiment, as defined by Definition 3, in Game i, and $\mathsf{adv}_i = \Pr[E_i] - \frac{1}{2}$.

Game 0. This game corresponds to the key indistinguishability experiment described in Section 4.1. Therefore

$$\Pr[E_0] = \frac{1}{2} + \mathsf{adv}^{\mathsf{key\text{-}ind}}_{PPSAKE} = \frac{1}{2} + \mathsf{adv}_0$$

Game 1. The challenger aborts the experiment and chooses $b' \in \{0,1\}$ uniformly at random if there exists an instance that accepts maliciously. In other words, in this game we make the same modifications as in the games performed during the entity authentication proof. Hence

$$\mathsf{adv}_0 \leq \mathsf{adv}_1 + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{PPSAKE}$$

Game 2. The challenger tries to guess which instance is targeted by the adversary. If the guess is wrong, the game is aborted. The number of instances is at most nq. Therefore

$$\mathsf{adv}_2 = \mathsf{adv}_1 \times \frac{1}{nq}$$

Game 3. Let π be the instance targeted by the adversary. We reduce the advantage of the adversary to win this game to the security of the function KDF used to compute the session key. That is, we rely upon the pseudo-randomness of the KDF function. This is possible if the key K is random. The genuine value of K is uniformly chosen at random by assumption. Then K is updated with update at most once per session on average. Therefore, when the u-th session starts, K has been updated at most $u-1$ times already. Therefore we must take into account the successive losses due to the key update with respect to the pseudo-randomness of update. Since there is at most q sessions per party (i.e., per original key K), this loss is at most $(q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}$. Hence we replace each function $\mathsf{update}(K) = \mathsf{PRF}(K, c)$ (keyed with a different key K throughout the, at most, $q-1$ successive sessions established, prior to that current session, by the same party that owns π) with truly random functions $\mathsf{G}^{\mathsf{update}}_0$, ..., $\mathsf{G}^{\mathsf{update}}_{q-2}$. Moreover, if an instance uses the same key $K = K_i$, $0 \leq i < q-1$, to key update, then we replace update with the corresponding random function $\mathsf{G}^{\mathsf{update}}_i$. Since, to that point, the key $K = K_{q-1}$ used to compute the session key is random, we reduce the ability of $\mathcal{A}$ to win to the security of KDF. Therefore

$$\mathsf{adv}_2 \leq \mathsf{adv}_3 + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}} + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}}$$

To that point the session key is random, therefore the adversary has no advantage in guessing $\pi.\mathsf{b}$. That is

$$\mathsf{adv}_3 = 0$$

Collecting all the probabilities from Game 0 to 3, we have that

$$\begin{aligned}
\mathsf{adv}^{\mathsf{key\text{-}ind}}_{PPSAKE} &= \mathsf{adv}_0 \\
&\leq \mathsf{adv}_1 + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{PPSAKE} \\
&= nq \times \mathsf{adv}_2 + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{PPSAKE} \\
&\leq nq \left(\mathsf{adv}_3 + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}} + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}} \right) + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{PPSAKE} \\
&= nq \left((q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}} + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}} \right) + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{PPSAKE}
\end{aligned}$$

A.3 Extended Proof for Privacy

Let E_i be the event that an adversary wins the privacy experiment, as defined by Definition 4, in Game i, and $\mathsf{adv}_i = \Pr[E_i] - \frac{1}{2}$.

Game 0. This game corresponds to the privacy experiment described in Section 4. Therefore

$$\Pr[E_0] = \frac{1}{2} + \mathsf{adv}^{\mathsf{privacy}}_{PPSAKE} = \frac{1}{2} + \mathsf{adv}_0$$

Game 1. In this game, we add an abort rule. The challenger aborts the experiment and chooses $b' \in \{0,1\}$ uniformly at random if there exists an instance that accepts maliciously. In other words, we make the same modifications as in the games performed during the entity authentication proof. Hence

$$\mathsf{adv}_0 \leq \mathsf{adv}_1 + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{PPSAKE}$$

At this point, we have excluded active adversaries. Moreover, for any instance π_i^s, there exists a unique instance π_j^t such that π_i^s and π_j^t have matching conversations. Therefore any accepting instance has an unique identified partner.

Game 2. The challenger tries to guess which instance is targeted by the adversary. If the guess is wrong, the game is aborted. The number of instances is at most $(n_{\mathsf{E}} + n_{\mathsf{S}}) \times q = nq$. Therefore

$$\mathsf{adv}_2 = \mathsf{adv}_1 \times \frac{1}{nq}$$

Game 3. The challenger aborts the experiment if the adversary succeeds in guessing the initial value of the derivation master key K used by the two instances π_i^s and π_j^t.

The rule aims at precluding the following attack. The adversary can try to guess the initial value of K as follows: from a guess value, it updates the value as many times as necessary in order to get the presumed value for the derivation master key used to compute the current session key. The adversary computes the presumed session key $\tilde{sk}$. Then, when π_i^s accepts, the adversary issues a Reveal query which yields the current session key sk. The adversary compares sk and $\tilde{sk}$ in order to verify if its guess for the initial value of K is correct. Once the initial value for K is found, the adversary computes the current value of the derivation master key. Once π_i^s accepts, it issues a $\mathsf{Corrupt}$ query to P_i and P_j (where $vid = P_i|P_j$ is the virtual identifier of the parent of either π_i^s or π_j^t, whichever is an $\mathsf{end\text{-}device}$ instance). This yields the derivation master key of P_i and P_j. The adversary compares these two keys with $\tilde{K}$ which indicates the secret bit.

Let κ be the bit length of the derivation master key. Since the initial value of the derivation master key is uniformly drawn at random, we have that

$$\mathsf{adv}_2 \leq \mathsf{adv}_3 + \frac{1}{2^\kappa}$$

Game 4. The challenger aborts the experiment if the adversary succeeds in correlating any value (besides id_A) exchanged during the current session with

any value exchanged during a previous session. The adversary can rely on two parameters: the identity value id_B, and the MAC tags.

The identity parameter is updated as $\mathsf{update}(K', id_B) = \mathsf{PRF}(K', id_B)$. The authentication master key is updated as $K' = \mathsf{update}(K') = \mathsf{PRF}(K', c)$. Since the initial value K' is random, we replace $\mathsf{update}(K') = \mathsf{PRF}(K', c)$ with a truly random function $\mathsf{F}_0^{\mathsf{update}}$. Moreover if an instance uses the same value K' to key update, we replace update with $\mathsf{F}_0^{\mathsf{update}}$.

When the u-th session starts, the authentication master key used to compute the current identity parameter has been updated at most $u-2$ times. Since there is at most q sessions per party, this means that the authentication master key used to compute the current identity parameter has been updated at most $q-2$ times. Therefore, we replace each function $\mathsf{update}(K') = \mathsf{PRF}(K', c)$ (keyed with a different key K' throughout the, at most, $q-1$ successive sessions established, prior to that current session, by the same parties that own π_i^s and π_j^t) with truly random functions $\mathsf{F}_0^{\mathsf{update}}$, ..., $\mathsf{F}_{q-3}^{\mathsf{update}}$. Moreover, if an instance uses the same key $K' = K'_i$, $0 \leq i < q-2$, to key update, then we replace update with the corresponding random function $\mathsf{F}_i^{\mathsf{update}}$. Hence, to that point, the key $K' = K'_{q-2}$ used to compute the identity parameter id_B is random. This implies a security loss at most $(q-2)\mathsf{adv}_{\mathsf{update}}^{\mathsf{prf}}$. To that point, id_B is computed with the function update keyed with a random key ($K' = K'_{q-2}$). Therefore, we rely on the pseudo-randomness of update in order to replace id_B with a random value (i.e., id_B is output by $\mathsf{F}_{q-2}^{\mathsf{update}}$). This adds a security loss $\mathsf{adv}_{\mathsf{update}}^{\mathsf{prf}}$. Moreover, the identity parameter exchanged in messages of a given session is either output by the update function (if $\phi = 0$) or uniformly chosen at random (if $\phi = 1$). Therefore, id_B in the current session can be replaced with a random value.

Regarding the MAC tags, when the u-th session starts, the authentication key has been updated $u-1$ times on average, and it is updated once on average during the current session (two different values for the authentication key are used during the current session if the latter completes successfully). Since there is at most q sessions, this means that the authentication key is updated $q-1$ times when the current session starts, and once again during the current session. Continuing the previous reasoning, this leads to a security loss $q \times \mathsf{adv}_{\mathsf{update}}^{\mathsf{prf}}$ with respect to the PRF-security of update.

Finally, we rely upon the pseudo-randomness of the MAC function. This implies a loss equal to $2 \times \mathsf{adv}_{\mathsf{Mac}}^{\mathsf{prf}}$ corresponding to the ability of an adversary $\mathcal{A}_3$ to distinguish the function Mac from a random function (because the MAC tags are output by the function Mac keyed with two different authentication master keys during the same session, when the latter is successful). Therefore, the MAC tags in the current session can be replaced with random values.

Therefore we have that

$$\mathsf{adv}_3 \leq \mathsf{adv}_4 + q \times \mathsf{adv}_{\mathsf{update}}^{\mathsf{prf}} + 2 \times \mathsf{adv}_{\mathsf{Mac}}^{\mathsf{prf}}$$

To that point, the identity value and the MAC tags that are exchanged during the current session are random and independent of all previous sessions. Therefore the adversary has no advantage in guessing which previous session the

current session is related to. In particular it has no advantage in guessing $vid.\mathsf{b}$ where vid is the virtual identifier of π_i^s if $\mathsf{type}(\pi_i^s) = \mathsf{end\text{-}device}$ or $\pi_i^s.\mathsf{pid} = vid$ if $\mathsf{type}(\pi_i^s) = \mathsf{server}$. Hence

$$\mathsf{adv}_4 = 0$$

Collecting all the probabilities from Game 0 to Game 4, we have that

$$\begin{aligned}
\mathsf{adv}^{\mathsf{privacy}}_{PPSAKE} &= \mathsf{adv}_0 \\
&\leq \mathsf{adv}_1 + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{PPSAKE} \\
&= nq \times \mathsf{adv}_2 + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{PPSAKE} \\
&\leq nq \left(\mathsf{adv}_3 + \frac{1}{2^\kappa} \right) + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{PPSAKE} \\
&\leq nq \left(\mathsf{adv}_4 + q \cdot \mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}} + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{Mac}} + \frac{1}{2^\kappa} \right) + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{PPSAKE} \\
&= nq \left(q \cdot \mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}} + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{Mac}} + 2^{-\kappa} \right) + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{PPSAKE}
\end{aligned}$$

B Aghili et al.'s Protocols

Our description of Aghili et al.'s protocols is mainly based on Section 6 complemented with Section C.2 of their paper [1]. The following notations are used in Figures 8 and 9.

The predicate "$x \in \mathsf{db.id}$" means that x is equal to one of the identity parameters $id_{B,t}$ stored in database db.

The value id'_B in m_A is the identity parameter corresponding to the authentication master key which verifies correctly τ_B in m_B (operation done by A).

The parameter *entry*, and each entry of the database db are of the form: $(K, (id_{B,j}, K'_j), (id_{B,j-1}, K'_{j-1}), (id_{B,j+1}, K'_{j+1}), r_{temp})$.

The parameters ϵ, K', and id'_B are set in the function $\mathsf{verif\text{-}entry}$ (which is also called by the function $\mathsf{find\text{-}entry}$).

The notation upd^r corresponds to the update of $r_{temp} = (r'', r')$ with r_B, and is defined as follows:

1. $r'' \leftarrow r'$
2. $r' \leftarrow r_B$

The notations kdf, upd_A, and upd_B are defined as follows:

- kdf corresponds to: $sk \leftarrow \mathsf{KDF}(K, r_A, r_B)$
- upd_A corresponds to
  1. $K \leftarrow \mathsf{update}(K)$
  2. $id_{B,j-1} \leftarrow id_{B,j}$
  3. $id_{B,j} \leftarrow id_{B,j+1}$
  4. $id_{B,j+1} \leftarrow \mathsf{update}(id_{B,j+1} \| K'_{j+1})$
  5. $K'_{j-1} \leftarrow K'_j$
  6. $K'_j \leftarrow K'_{j+1}$

7. $K'_{j+1} \leftarrow \mathsf{update}(K'_{j+1})$

– upd_B corresponds to
1. $K \leftarrow \mathsf{update}(K)$
2. $id_B \leftarrow \mathsf{update}(id_B \| K')$
3. $K' \leftarrow \mathsf{update}(K')$

The function verif-entry takes as input an entry $entry \in \mathsf{db}$, and a message $m_B = x \| r_B \| \tau_B$ (we assume that the other values used in verif-entry are "global" parameters). It outputs `true` if $entry$ allows verifying correctly m_B. The function verif-entry is described with the pseudo-code given in Figure 10. The line "$r_A \leftarrow \emptyset$" in Figure 9 indicates that r_A is not involved in the subsequent calls to the function verif-entry (when the end-device is initiator). For instance, the first "`if`" statement in Figure 10 corresponds then to the predicate "$\mathsf{Vrf}(K'_j, id_{B,j} \| id_A \| r_B, \tau_B) = \mathtt{true}$".

The function find-entry takes as input a message $m_B = x \| r_B \| \tau_B$, and outputs either an entry $entry \in \mathsf{db}$ or $\emptyset$. The function find-entry is described with the pseudo-code given in Figure 11.

A (id_A, db)		B (id_B, K, K', ϕ)
$r_A \xleftarrow{\$} \{0,1\}^\lambda$	$\xrightarrow{id_A \| r_A}$	

		$r_B \xleftarrow{\$} \{0,1\}^\lambda$
		$\tau_B \leftarrow \mathsf{Mac}(K', id_B \| id_A \| r_B \| r_A)$
		
		`if` $(\phi = 0)$
		  $m_B \leftarrow id_B \| r_B \| \tau_B$
		  $\phi \leftarrow 1$
		`else if` $(\phi = 1)$
		  $r_\alpha \xleftarrow{\$} \{0,1\}^\lambda$
		  $m_B \leftarrow r_\alpha \| r_B \| \tau_B$

	$\xleftarrow{m_B}$	

// $m_B = x \| r_B \| \tau_B$
`if` $(x \in \mathsf{db.id})$
  $entry \leftarrow$ get corresponding entry
  `if` $(r_B \in r_{temp})$
  abort
  `if` $(\mathsf{verif\text{-}entry}(entry, m_B) = \mathtt{false})$
  abort
`else`
  $entry \leftarrow \mathsf{find\text{-}entry}(m_B)$
  `if` $(entry = \emptyset)$
  abort
  `if` $(r_B \in r_{temp})$
  abort
 	
upd^r
$\tau_A \leftarrow \mathsf{Mac}(K', \epsilon \| id_A \| id'_B \| r_A \| r_B)$
$m_A \leftarrow \epsilon \| id'_B \| \tau_A$		

	$\xrightarrow{m_A}$	

		`if` $(\mathsf{Vrf}(K', \epsilon \| id_A \| id_B \| r_A \| r_B, \tau_A) = \mathtt{false})$
		  abort
		`if` $(\epsilon = 1)$
		  upd_B
		
		kdf; upd_B
		$\phi \leftarrow 0$
		$\tau'_B \leftarrow \mathsf{Mac}(K', r_B \| r_A)$
		$m'_B \leftarrow id_B \| \tau'_B$

	$\xleftarrow{m'_B}$	

`if` $(\epsilon = 0)$
  $K' \leftarrow K'_j$
  `if` $(\mathsf{Vrf}(K', r_B \| r_A, \tau'_B) = \mathtt{false})$
  abort
`else if` $(\epsilon = 1)$
  $K' \leftarrow K'_{j+1}$
  `if` $(\mathsf{Vrf}(K', r_B \| r_A, \tau'_B) = \mathtt{false})$
  abort
  kdf; upd_A
 	
$\tau'_A \leftarrow \mathsf{Mac}(K', r_A \| r_B)$
$m'_A \leftarrow id_B \| \tau'_A$		

	$\xrightarrow{m'_A}$	
		`if` $(\mathsf{Vrf}(K', r_A \| r_B, \tau'_A) = \mathtt{false})$ abort

Fig. 8: Aghili et al.'s protocol when the server is initiator

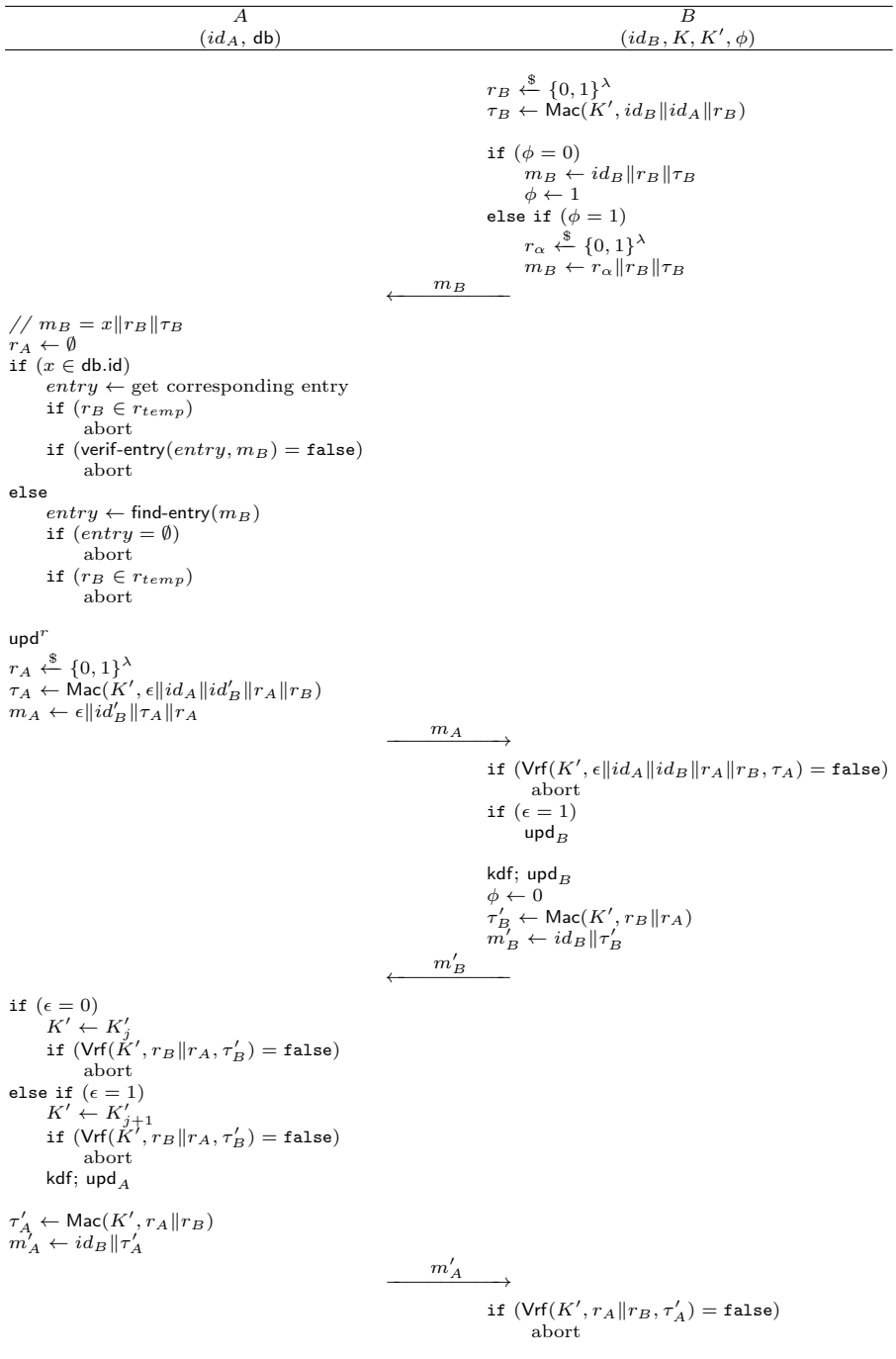

Fig. 9: Aghili et al.'s protocol when the end-device is initiator

```
if (Vrf(K'_j, id_{B,j} || id_A || r_B || r_A, τ_B) = true)
    δ_AB ← 0
    K' ← K'_j
    id'_B ← id_{B,j}
    kdf
    upd_A
    ϵ ← 0
    return true
else if (Vrf(K'_{j-1}, id_{B,j-1} || id_A || r_B || r_A, τ_B) = true)
    δ_AB ← 1
    K' ← K'_{j-1}
    id'_B ← id_{B,j-1}
    ϵ ← 1
    return true
else if (Vrf(K'_{j+1}, id_{B,j+1} || id_A || r_B || r_A, τ_B) = true)
    δ_AB ← −1
    K' ← K'_{j+1}
    id'_B ← id_{B,j+1}
    upd_A
    kdf
    upd_A
    ϵ ← 0
    return true
else
    return false
```

Fig. 10: Pseudo-code of function verif-entry in Aghili et al.'s protocols

```
foreach entry ∈ db
    if (verif-entry(entry, m_B) = true)
        return entry
return ∅
```

Fig. 11: Pseudo-code of function find-entry in Aghili et al.'s protocols