Symmetric-Key Authenticated Key Exchange (SAKE) with Perfect Forward Secrecy

Gildas Avoine[1,2], Sébastien Canard[3], and Loïc Ferreira[1,3(✉)]

[1] Univ Rennes, INSA Rennes, CNRS, IRISA, Rennes, France
gildas.avoine@irisa.fr
[2] Institut Universitaire de France, Paris, France
[3] Orange Labs, Applied Crypto Group, Caen, France
{sebastien.canard,loic.ferreira}@orange.com

Abstract. Key exchange protocols in the asymmetric-key setting are known to provide stronger security properties than protocols in symmetric-key cryptography. In particular, they can provide *perfect forward secrecy*, as illustrated by key exchange protocols based on the Diffie-Hellman scheme. However public-key algorithms are too heavy for low-resource devices, which can then not benefit from forward secrecy. In this paper, we describe a scheme that solves this issue. Using a shrewd resynchronisation technique, we propose an authenticated key exchange protocol in the symmetric-key setting that guarantees perfect forward secrecy. We prove that the protocol is sound, and provide a formal proof of its security.

Keywords: Authenticated key agreement · Symmetric-key cryptography · Perfect forward secrecy · Key-evolving

1 Introduction

An authenticated key exchange (AKE) protocol executed between two parties aims at authenticating the parties, and computing a fresh shared session key. Well-known two-party authenticated key exchange protocols make use of digital signatures to provide authentication, and apply the Diffie-Hellman (DH) scheme [20] to compute a shared session key. However, such protocols are too heavy for low-resource devices. More suited protocols, solely based on symmetric-key functions, have been proposed (e.g., [12,16,23,26,29,30,33,34] to cite a few), including widely deployed ones (e.g., in 3G/UMTS [2] and 4G/LTE [3]). Such symmetric-key protocols are needed in various applications, ranging from Wireless Sensor Networks (WSNs), Radio Frequency Identification (RFID) tags, smart cards, Controller Area Networks (CANs) for vehicular systems, smart home, up to industrial Internet of Things (IoT). Yet, existing symmetric-key based protocols lack a fundamental security property usually provided by the

S. Jarecki (Ed.): CT-RSA 2020, LNCS 12006, pp. 199–224, 2020.
https://doi.org/10.1007/978-3-030-40186-3_10

DH scheme: *perfect forward secrecy* (PFS) [21,24]. PFS is a very strong form of long-term security which, informally, guarantees that future disclosures of some long-term secret keys do not compromise past session keys. It is widely accepted that PFS can only be provided by asymmetric schemes. Indeed, in protocols based on symmetric-key functions, the two parties must share a long-term symmetric key (which the session keys are computed from). Therefore the disclosure of this static long-term key allows an adversary to compute all the past (and future) session keys. In this paper, we introduce an AKE protocol in the *symmetric-key* setting, and, yet, that does guarantee PFS.

1.1 Related Work

Symmetric-key based protocols do not provide the same security guarantees as those based on asymmetric algorithms. In particular, they do not guarantee forward secrecy. Nonetheless, (a few) attempts aim at proposing symmetric-key protocols that incorporate forward secrecy, as illustrated by the following related work.

Dousti and Jalili [22] describe a key exchange protocol where the shared master key is updated based on time. Their protocol requires perfect synchronicity between the parties otherwise this leads to two main consequences. Firstly, in order to handle the key exchange messages, the parties may use different values of the master key corresponding to consecutive epochs, which causes the session to abort. Secondly, this allows an adversary to trivially break forward secrecy. Once a party deems the protocol run is correct and the session key can be safely used (i.e., once the party "accepts"), the adversary corrupts its partner (which still owns the previous, not updated yet, master key), and computes the current session key. Furthermore, achieving perfect time synchronisation may be quite complex in any context, in particular for low-resource devices. Contrary to Dousti et al., the protocol we propose explicitly deals with the issue of updating the master keys at both parties without requiring any additional functionality (such as a synchronised clock).

In the RFID field, the protocol proposed by Le, Burmester, and de Medeiros [28] aims at authenticating a tag to a server, and at computing a session key in order to establish a secure channel (which they do not describe). The master key is updated throughout the protocol run. To deal with the possible desynchronisation between the reader and the tag, the server keeps two consecutive values of the key: the current and the previous one. If the tag does not update its master key (which happens when the last message is dropped), the server is able to catch up during the next session. This implies that, in case of desynchronisation, the server computes the session key from the updated master key, whereas the tag still stores the previous value. Hence, an adversary that corrupts the tag can compute the previous session key with respect to the server. In fact, since the server always keeps the previous value of the master key, together with the current one, the scheme is intrinsically insecure in strong security models (i.e., models that allow the adversary to corrupt any of the partners, once the targeted party accepts). Yet, Le et al. analyse their protocol in a model where

any server corruption is forbidden, and corrupting a tag is allowed only once it accepts. In our scheme, one of the parties also keeps in memory (a few) samples of a master key corresponding to different epochs (including a previous one). Yet the disclosure of all these values does *not* compromise past session keys. Furthermore, the (strong) security model we use allows the adversary to corrupt either partner as soon as the targeted party accepts.

Brier and Peyrin [17] propose a forward secret key derivation scheme in a client-server setting, that aims at improving a previous proposal [7]. In addition to forward secrecy, another constraint is that the amount of calculation to compute the master key (directly used as encryption key) on the server side must be low. Their solution implies the storage, on the client side, of several keys in parallel and to use a (short) counter, which is involved in the keys update. The keys belong to a tree whose each leaf (key) is derived from the previous one and the counter. The client must send the counter with the encrypted message for the server to be able to compute the corresponding key. The main drawback of this scheme is that the number of possible encryption keys is reduced. Increasing that limit implies increasing the counter size and the number of keys stored in parallel on the client side. Moreover, Brier et al. (as well as [7]) focus on forward secrecy with respect to the client only. The server is deemed as incorruptible, and is supposed to compute an encryption key only upon reception of a client's message (the secure channel is unidirectional, and the server does not need to send encrypted messages to the client). Therefore, the scheme does not need to deal with the issue of *both* parties being in sync (with respect to the key computation), and providing forward secrecy. In addition, the purpose of Brier et al. (as well as [7]) is not to provide mutual authentication. More generally sending additional information in order to resynchronise (such as a sufficiently large counter) is a simple (and inefficient) way to build a forward secret protocol. But this yields several drawbacks. Firstly, the size of such a counter must be large enough in order to avoid any exhaustion. Secondly, sending the counter (at least periodically) is necessary for the two parties to resynchronise, which consumes bandwidth. Thirdly, resynchronisation may imply multiple updates of the master keys at once (the scheme of Brier et al. and [7] aims at limiting that amount of calculation, but it leads to a narrowed number of possible encryption keys). Our scheme avoids all these drawbacks.

The more general question of forward security in symmetric cryptography has been also investigated by Bellare and Yee [14]. They propose formal definitions and practical constructions of forward secure primitives (e.g., MAC, symmetric encryption algorithm). Their constructions protect against decryption of past messages, or antedated forgeries of messages (i.e., previously authenticated messages are made untrustworthy). Their algorithms are based on key-evolving schemes [10]. Nonetheless, Bellare et al. consider only algorithms (but not protocols) and they do not deal with the issue of synchronising the evolution of the shared key at *both* parties. That is, they propose out-of-context (non-interactive) solutions with respect to our purpose.

Abdalla and Bellare [4] investigate a related question which is "re-keying". Their formal analysis show that appropriate re-keying techniques "increase" the lifetime of a key. They consider re-keying in the context of symmetric encryption (in order to thwart attacks based on the ability to get lots of encrypted messages under the same key), and forward security (in order to protect past keys). Yet, they confine their analysis to algorithms and not protocols. Hence, as Bellare et al. [14], they do not treat the synchronisation issues that arise from evolving a shared symmetric key.

The Signal messaging protocol [1] uses a key derivation scheme called "double ratchet algorithm" [31]. This scheme combines a DH based mechanism with a symmetric key-evolving mechanism (based on a one-way function). The first mechanism provides an asymmetric ratchet, whereas the second provides a symmetric ratchet. The asymmetric ratchet is applied when a fresh DH share is received (included in an application message) from the peer. The symmetric ratchet is applied when a party wants to send several successive messages without new incoming message from its partner. Thanks to the DH scheme, the asymmetric ratchet is supposed to provide forward secrecy.[1] Regarding the symmetric ratchet, each party is compelled to store the decryption keys of the not yet received messages. This is due to the asynchronous nature of the Signal protocol. Therefore, the symmetric ratchet in Signal does not provide forward secrecy, as stated in their security analysis by Cohn-Gordon, Cremers, Dowling, Garratt, and Stebila [19]: "*old but unused receiving keys are stored at the peer for an implementation dependent length of time, trading off forward security for transparent handling of outdated messages. This of course weakens the forward secrecy of the keys*". Consequently, Cohn-Gordon et al. choose not to model this weakened property. In turn, Alwen, Coretti, and Dodis [6] incorporate the latter in the security analysis of their "generalised Signal protocol". But the crucial difference in their notion of forward security is that, as soon as the receiver is compromised, no more security can be provided. On the contrary, we tackle the synchronisation issue, and solve it in our protocol. The security model we use captures forward secrecy and allows corrupting a party and its partner as soon as the targeted party "accepts" (i.e., deems the session key can be safely used). With regard to Signal, our protocol can be compared to the asymmetric ratchet (in synchronous mode), and yet does not implement asymmetric functions.

We stress that the goals of several of the aforementioned protocols are not the same as ours. Nonetheless, the small number of existing symmetric-key protocols that provide forward secrecy, and the lukewarm security level they achieve illustrate that combining symmetric-key cryptography and (a strong form of) forward secrecy is a non-trivial task.

1.2 Contributions

We describe the SAKE protocol, a two-party authenticated key exchange protocol in the *symmetric-key* setting with the following characteristics.

[1] In Signal, the DH exchanges can be asynchronous. This impairs the forward secrecy property usually ensured by this scheme.

- It guarantees forward secrecy.
- It is self-synchronising. That is, after a correct and complete session (and whatever the internal state of the parties prior to the session), the two parties involved in the protocol run share a new session key, and their internal state is updated and synchronised.
- It allows establishing an (virtually) unlimited number of sessions (as opposite to symmetric-key protocols that make use of a predefined list of master keys, each being used once only).
- The amount of calculation done by both parties in a single protocol run is strictly bounded. In particular we avoid the need of sending additional information in order to resynchronise, such as a (sufficiently large) counter that keeps track of the evolution of the master keys, and the subsequent drawbacks: periodically doing a great amount of computations at once (when resynchronisation is necessary), and consuming bandwidth (to transmit the additional data).

In addition, we provide a formal security proof for SAKE. We also present a complementary mode of SAKE (that we call SAKE-AM) which is an "aggressive mode" of the protocol. This mode inverts the role of the initiator and the responder in terms of calculations (in SAKE, the initiator performs – at most – two additional MAC computations compared to the responder). Using SAKE and SAKE-AM together results in an implementation (gathering all the aforementioned properties, starting with the forward secrecy property) that allows any party to be either initiator or responder of a session, and such that the smallest amount of calculation is always done by the same party. This is particularly convenient in the context of a set of (low-resource) end-devices communicating with a central server. In such a case, the end-device supports the smallest amount of calculation, whereas either the server or the end-device can initiate a session.

1.3 Our Approach

Key Concepts. The authenticated key exchange protocol we propose is *solely* based on symmetric-key functions. Not only does it provide mutual authentication and key agreement, but it guarantees perfect forward secrecy. We attain this very strong form of long-term security by using a key-evolving scheme. As soon as two parties make a shared (symmetric) key evolve, a synchronisation problem arises. We provide a simple and efficient solution to this issue. We require using neither a clock, nor an additional resynchronising procedure. Our solution is based on a second (independent) chain of master keys. These keys allow tracking the evolution of the internal state, and resynchronising the parties if necessary. The parties authenticate each other prior to updating their master keys. Hence the possible gap is bounded (as we prove it), and each party is always able to catch up in case of desynchronisation (of course, if the session is correct and complete). Mutual authentication, key exchange (with forward secrecy), and resynchronisation are done in the continuity of the protocol run.

Our protocol is based on two symmetric master keys: a derivation master key K and an authentication master key K'. The protocol makes use of symmetric-key functions only. Each pair of parties (A, B) shares distinct master keys. The main lines of the protocol are as follows. The two parties exchange pseudo-random values r_A, r_B which are used to

- authenticate each other: each party sends back the value it has received in a message that is MAC-ed with the authentication master key K'. For instance, if B receives r_A it replies with $r_B \| \tau_B$ where $\tau_B = \mathsf{Mac}(K', B \| A \| r_B \| r_A)$.
- Compute a session key: a pseudo-random function KDF is keyed with the derivation master key K and uses the pseudo-random values as input. That is, $sk \leftarrow \mathsf{KDF}(K, f(r_A, r_B))$. $f(r_A, r_B)$ is deliberately left undefined, and designates an operation between r_A and r_B such as the concatenation or the bitwise addition.

Providing Forward Secrecy. The shared key K is used to compute the session keys. If this key remains unchanged throughout all protocol runs, its disclosure allows computing all past (and future) session keys. To solve this issue we apply a key-evolving technique. We update the master key such that a previous master key cannot be computed from an updated one. Each of the two parties involved in a session updates its own copy of the derivation master key K with a non-invertible function update: $K \leftarrow \mathsf{update}(K)$. Hence this protects past sessions in case the (current value of) master key K is revealed. Each party authenticates its peer prior to updating the derivation master key. If the master key is updated throughout the session, it may happen that one of the two involved parties update its master key whereas the other does not. This leads to a *synchronisation problem.*

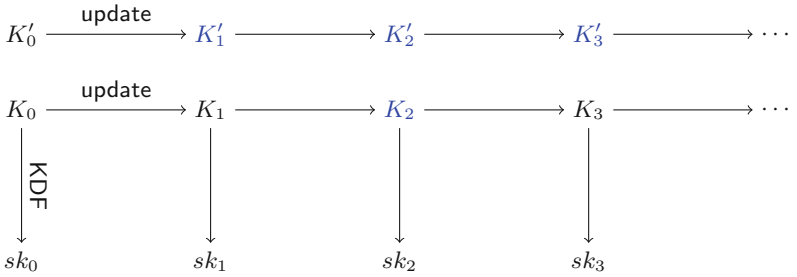


Fig. 1. Master key chains in SAKE. At epoch j, the initiator stores four keys: $K = K_j$, and K'_{j-1}, K'_j, K'_{j+1}. The responder stores two keys: $K = K_j$ and $K' = K'_j$. (Illustration with $j = 2$)

The Synchronisation Problem. If two parties use a different key K, they are obviously not able to compute a shared session key. Hence they must resynchronise first. More fundamentally, if a party initiates a session with some derivation

master key K, and its partner stores a master key corresponding to an earlier epoch, then an adversary that corrupts the partner can compute past session keys with respect to the initiator, hence trivially break forward secrecy. Therefore, it is of paramount importance that the parties know if the master key of its partner has actually been updated. We provide a solution to both issues in the continuity of a *single* session. In particular, no extra procedure is needed in order for a desynchronised party to catch up. We avoid the need of sending additional information in order to resynchronise, such as a (sufficiently large) counter that keeps track of the evolution of the master keys, and the subsequent drawbacks: periodically doing a great amount of computations at once (when resynchronisation is necessary), and consuming bandwidth (to transmit the additional data). We base our solution on the second master key K' used to authenticate the messages exchanged during a session. The solution is to update K' at the same time as K (see Fig. 1). Therefore the evolution of K' follows that of K. The party that receives the first authenticated message uses the MAC tag to learn which epoch the sender belongs to. Of course, K' can also be desynchronised in the same way as K. This is why, whereas one party (responder B) stores only one sample of the key K', the other party (initiator A) stores several samples of the authentication master key K' corresponding to several consecutive epochs. We *prove* that only three keys K'_{j+1}, K'_j, K'_{j-1}, corresponding respectively to the next, the current, and the previous epochs, are sufficient in order for A and B to resynchronise. The initiator (A) is the one able to deal with the synchronisation issue, and consequently tells B how to behave. Each party "accepts" only after it has received a confirmation (final MAC-ed messages) that its partner has already updated its own master keys. In such a case, the party ending in accepting state deems that the fresh session key can be safely used. Otherwise (in particular when the parties are desynchronised), the session key is discarded.

Since two independent master keys are used (authentication and session key derivation), one can safely maintain a copy of K' corresponding to an earlier epoch (K'_{j-1}) without risk of threatening forward secrecy. Only one sample of the derivation master key K is kept: the most up-to-date.

1.4 Outline of the Paper

In Sect. 2 we detail the security model used to analyse the protocol we propose. Our authenticated key exchange protocol in symmetric-key setting with forward secrecy is described in Sect. 3. In Sect. 4, we investigate the feasibility of a variant based on our protocol. Formal proofs of soundness and security for the main protocol are presented in Sect. 5. The differences between our approach and the DH scheme are highlighted in Sect. 6. Finally, we conclude in Sect. 7.

2 Security Model

Before describing our symmetric-key protocol in Sect. 3 (which is self-sufficient and contains all the specifics required to understand the protocol), we present in this section the security model that we employ to formally prove its security.

In a nutshell, we use the model for authenticated key exchange protocols described by Brzuska, Jacobsen, and Stebila [18]. This model incorporates all the features that are usually considered when analysing key agreement protocols in the public-key setting (e.g., DH-based protocols with signature). In this model, the adversary has full control over the communication network. It can forward, alter, drop any message exchanged by honest parties, or insert new messages. Brzuska et al.'s model then captures adaptive corruptions but also perfect forward secrecy. This appears in the definition of the security experiment.

2.1 Execution Environment

In this section, we present the security model for authenticated key exchange protocols described by Brzuska et al. [18], and reuse the corresponding notation.

Parties. A two-party protocol is carried out by a set of parties $\mathcal{P} = \{P_0, \ldots, P_{n-1}\}$. Each party P_i has an associated long-term key ltk. Each pair of parties shares a distinct key ltk.[2]

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 (we elaborate more on this in Sect. 6). 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 of the party. In addition, each instance maintains the following state specific to the session.

- ρ: the role $\rho \in \{\mathsf{init}, \mathsf{resp}\}$ 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 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:

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

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

[2] Note that ltk can be a set of master keys (e.g., each one used by the party for a different purpose).

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

- $\mathsf{NewSession}(P_i, \rho, \mathsf{pid})$: this query creates a new instance π_i^s at party P_i, having role ρ, and intended partner pid.
- $\mathsf{Send}(\pi_i^s, m)$: 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.
- $\mathsf{Corrupt}(P_i)$: this query returns the long-term key $P_i.\mathsf{ltk}$ of 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 $\nu = +\infty$.
- $\mathsf{Reveal}(\pi_i^s)$: this query returns the session key $\pi_i^s.\mathsf{sk}$, and $\pi_i^s.\kappa$ is set to $\texttt{revealed}$.
- $\mathsf{Test}(\pi_i^s)$: 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, where $sk_1 = \pi_i^s.\mathsf{sk}$. The key sk_b is called the Test-*challenge*.

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

Definition 2 (Freshness). *An instance π_i^s is said to be* fresh *with intended partner P_j, if*

(a) $\pi_i^s.\alpha = \texttt{accepted}$ and $\pi_i^s.\mathsf{pid} = P_j$ when $\mathcal{A}$ issues its ν_0-th query,
(b) $\pi_i^s.\kappa \neq \texttt{revealed}$ and P_i is ν-corrupted with $\nu_0 < \nu$, and
(c) 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 $\nu_0 < \nu'$.

Note that the notion of freshness incorporates a requirement for forward secrecy.

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

Definition 3 (Entity Authentication (EA)). *An instance π_i^s of a protocol Π is said to have* accepted maliciously *in the AKE security experiment with intended partner P_j, if*

(a) $\pi_i^s.\alpha = \texttt{accepted}$ and $\pi_i^s.\mathsf{pid} = P_j$ when $\mathcal{A}$ issues its ν_0-th query,
(b) P_i and P_j are ν- and ν'-corrupted with $\nu_0 < \nu, \nu'$, and
(c) 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:

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

Definition 4 (Key Indistinguishability). *An adversary $\mathcal{A}$ against a protocol Π, that issues its* Test*-query to instance π_i^s during the AKE security experiment,* answers the Test-challenge correctly *if it terminates with output b', such that*

(a) π_i^s is fresh with some intended partner P_j, and
(b) $\pi_i^s.\mathsf{b} = b'$.

The adversary's advantage is defined as

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

Definitions 3 and 4 allow the adversary to corrupt an instance involved in the security experiment (once the targeted instance has accepted, in order to exclude trivial attacks). Therefore, protocols secure with respect to Definition 5 below provide *perfect forward secrecy*. Note that we do not allow the targeted instance to be corrupted before it accepts. This security model does not capture key-compromise impersonation attacks (KCI) [15] since that would allow trivially breaking key exchange protocols solely based on shared symmetric keys.

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

2.2 Security Definitions of SAKE's Building Blocks

In this section, we recall the definitions of the main security notions we use in our results. The security definition of a pseudo-random function is taken from Bellare, Desai, Jokipii, and Rogaway [9], and that of a MAC strongly unforgeable under chosen-message attacks from Bellare and Namprempre [11]. We recall also the definition of matching conversations initially proposed by Bellare and Rogaway [12], and modified by Jager, Kohlar, Schäge, and Schwenk [27].

Secure PRF. A *pseudo-random function* (PRF) F is a deterministic algorithm which given a key $K \in \{0,1\}^\lambda$ and a bit string $x \in \{0,1\}^*$ outputs a string $y = F(K,x) \in \{0,1\}^\gamma$ (with γ being polynomial in λ). Let $Func$ be the set of all functions of domain $\{0,1\}^*$ and range $\{0,1\}^\gamma$. The security of a PRF is defined with the following experiment between a challenger and an adversary $\mathcal{A}$:

1. The challenger samples $K \xleftarrow{\$} \{0,1\}^\lambda$, $G \xleftarrow{\$} Func$, and $b \xleftarrow{\$} \{0,1\}$ uniformly at random.
2. The adversary may adaptively query values x to the challenger. The challenger replies to each query with either $y = F(K,x)$ if $b = 1$, or $y = G(x)$ if $b = 0$.
3. Finally, the adversary outputs its guess $b' \in \{0,1\}$ of b.

The adversary's advantage is defined as

$$\mathsf{adv}_F^{\mathsf{prf}}(\mathcal{A}) = \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

Definition 6 (Secure PRF). *A function* $F{:}\{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\gamma$ *is said to be a* secure pseudo-random function (PRF) *if, for all probabilistic polynomial time adversary* $\mathcal{A}$, $\mathsf{adv}_F^{\mathsf{prf}}(\mathcal{A})$ *is a negligible function in* λ.

Secure MAC. A message authentication code (MAC) consists of two algorithms $(\mathsf{Mac}, \mathsf{Vrf})$. The tagging algorithm Mac takes as input a key $K \in \{0,1\}^k$ and a message $m \in \{0,1\}^*$ and returns a tag $\tau \in \{0,1\}^\gamma$ (with γ being polynomial in k). The verification algorithm Vrf takes as input the key K, a message m, and a candidate tag τ for m. It outputs 1 if τ is a valid tag on message m with respect to K. Otherwise, it returns 0. The notion of *strong unforgeability under chosen-message attacks* (SUF-CMA) for a MAC $G = (\mathsf{Mac}, \mathsf{Vrf})$ is defined with the following experiment between a challenger and an adversary $\mathcal{A}$:

1. The challenger samples $K \xleftarrow{\$} \{0,1\}^k$, and sets $S \leftarrow \emptyset$.
2. The adversary may adaptively query values m to the challenger. The challenger replies to each query with $\tau = \mathsf{Mac}(K, m)$ and records (m, τ): $S \leftarrow S \cup \{(m, \tau)\}$.
   In addition, the adversary may adaptively query values (m', τ') to the challenger. The challenger replies to each query with $\mathsf{Vrf}(K, m', \tau')$.
3. Finally, the adversary sends (m^*, τ^*) to the challenger.

The adversary's advantage is defined as

$$\mathsf{adv}_G^{\mathsf{suf\text{-}cma}}(\mathcal{A}) = \Pr[\mathsf{Vrf}(K, m^*, \tau^*) = 1 \wedge (m^*, \tau^*) \notin S].$$

Definition 7 (SUF-CMA). *A message authentication code* $G = (\mathsf{Mac}, \mathsf{Vrf})$ *with* $\mathsf{Mac}{:}\{0,1\}^k \times \{0,1\}^* \to \{0,1\}^\gamma$ *is said to be* strongly unforgeable under chosen-message attacks (SUF-CMA) *if, for all probabilistic polynomial time adversary* $\mathcal{A}$, $\mathsf{adv}_G^{\mathsf{suf\text{-}cma}}(\mathcal{A})$ *is a negligible function in* k.

Matching Conversations. Let $T_{i,s}$ be the sequence of all (valid) messages sent and received by an instance π_i^s in chronological order. For two transcripts $T_{i,s}$ and $T_{j,t}$, we say that $T_{i,s}$ is a prefix of $T_{j,t}$ if $T_{i,s}$ contains at least one message, and the messages in $T_{i,s}$ are identical to the first $|T_{i,s}|$ messages of $T_{j,t}$.

Definition 8 (Matching Conversations). *We say that* π_i^s *has a* matching conversation *to* π_j^t, *if*

- π_i^s *has sent all protocol messages and* $T_{j,t}$ *is a prefix of* $T_{i,s}$, *or*
- π_j^t *has sent all protocol messages and* $T_{i,s} = T_{j,t}$.

3 Our Symmetric-Key AKE Protocol with Perfect Forward Secrecy

In this section we describe our main protocol. Although all the calculations are based on shared master keys, forward secrecy is guaranteed by using a key-evolving scheme. More precisely, we use two types of keys: one to compute the session keys, the other to authenticate messages and resynchronise when necessary. This second type of keys allows tracking the master keys evolution, and limit the gap (in terms of keys update) between both parties. Mutual authentication, key exchange, and synchronised update of the master keys are done in the same session.

3.1 Description of the Protocol

The protocol is depicted by Fig. 2. The parameter δ_{AB} computed by A corresponds to the gap between A and B with respect to the evolution of the master keys. We prove that $\delta_{AB} \in \{-1, 0, 1\}$ (see Sect. 5.1). That is, A can only be either *one step* behind, or in sync, or *one step* ahead to B. During a session, A uses the keys K'_j, K'_{j-1}, K'_{j+1} (by order of likelihood) and the first message (m_B) sent by B to learn δ_{AB}. The message m_B is computed with the current value K' of B. Therefore m_B indicates the current synchronisation state of B. Then A informs B. One bit ϵ is enough (message m_A) because B takes two behaviours only: if $\delta_{AB} \in \{-1, 0\}$ ($\epsilon = 0$), and if $\delta_{AB} = 1$ ($\epsilon = 1$). A and B behave as follows.

- If A is in sync with B ($\delta_{AB} = 0$), A computes the new session key, and updates its master keys. Then, upon reception of m_A, B does the same.
- If A is in advance ($\delta_{AB} = 1$), A waits for B to resynchronise (i.e., B updates its master keys a first time), and to proceed with the regular operations (i.e., B computes the new session key, and updates its master keys a second time). Then, once A receives a confirmation that B is synchronised (message τ'_B), A performs the regular operations as well (session key computation, master keys update). Since A waits for B to resynchronise before proceeding, the gap between the parties is *bounded* (as proved in Sect. 5.1).
- If A is late ($\delta_{AB} = -1$), it resynchronises (i.e., it updates its master keys a first time), and then performs the regular operations (session key computation, master keys update). Then (upon reception of message m_A), B applies the regular operations.

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) they are synchronised (which stems in particular from the fact that the gap between A and B is bounded, i.e., $|\delta_{AB}| \leq 1$), and (iii) they share a new session key. In other words, the protocol is *self-synchronising*.

Before the first session between A and B, the master keys are initialised as follows:

1. K and K' are uniformly chosen at random.
2. $K'_{j-1} \leftarrow \perp$
3. $K'_j \leftarrow K'$
4. $K'_{j+1} \leftarrow \mathsf{update}(K')$

Since K'_{j+1} and K'_j can be computed from K'_{j-1}, it is possible to store only K'_{j-1}, and to compute the two other keys when necessary during the session. Then, with respect to the security model presented in Sect. 2, the long-term key of A and B corresponds respectively to $A.\mathsf{ltk} = (K, K'_{j-1})$ and $B.\mathsf{ltk} = (K, K')$.

Although this does not appear explicitly in Fig. 2, a party aborts the session if it receives a message computed with an invalid identity. For the responder B an invalid identity corresponds to an initiator party A it does not share master keys with. For an initiator A, the particular case $B = A$, among other possibilities, yields an error (each party must have a distinct identity).

Number of Rounds. The session can be reduced from five to four messages in some cases. Indeed, regarding the synchronisation state, in two cases (when $\delta_{AB} \in \{-1, 0\}$, that is $\epsilon = 0$), A and B are synchronised, and share a session key once B has received message m_A and executed the subsequent operations. Therefore, in such a case, the session can end upon reception of message τ'_B by A. More precisely

- if $\delta_{AB} = 1$ ($\epsilon = 1$), then A accepts upon reception of τ'_B, and B accepts upon reception of τ'_A;
- if $\delta_{AB} \in \{-1, 0\}$ ($\epsilon = 0$), then A accepts upon reception of τ'_B, and B accepts upon reception of m_A.

Each message of the protocol fulfills a specific task: party authentication, detecting desynchronisation, and then catching up. This eventually results in the forward secrecy property being ensured. Removing one message yields an attack, as shown by any of the numerous alternative versions we have analysed. Although we do not formally prove it, we do think that the figure of five rounds is the least achievable in order to take into account all cases.

3.2 Notation

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

- kdf corresponds to: $sk \leftarrow \mathsf{KDF}(K, f(r_A, r_B))$
- 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. $K' \leftarrow \mathsf{update}(K')$

A $(K, K'_{j+1}, K'_j, K'_{j-1})$		B (K, K')
$r_A \xleftarrow{\$} \{0,1\}^\lambda$	$\xrightarrow{A\|r_A}$	
		$r_B \xleftarrow{\$} \{0,1\}^\lambda$ $\tau_B \leftarrow \mathsf{Mac}(K', B\|A\|r_B\|r_A)$ $m_B \leftarrow r_B\|\tau_B$
	$\xleftarrow{m_B}$	
if $(\mathsf{Vrf}(K'_j, B\|A\|r_B\|r_A, \tau_B) = \mathtt{true})$ $\delta_{AB} \leftarrow 0$ $K' \leftarrow K'_j$; kdf; upd_A; $\epsilon \leftarrow 0$ **else if** $(\mathsf{Vrf}(K'_{j-1}, B\|A\|r_B\|r_A, \tau_B) = \mathtt{true})$ $\delta_{AB} \leftarrow 1$ $K' \leftarrow K'_{j-1}$; $\epsilon \leftarrow 1$ **else if** $(\mathsf{Vrf}(K'_{j+1}, B\|A\|r_B\|r_A, \tau_B) = \mathtt{true})$ $\delta_{AB} \leftarrow -1$ $K' \leftarrow K'_{j+1}$; upd_A; kdf; upd_A; $\epsilon \leftarrow 0$ **else** abort		
$\tau_A \leftarrow \mathsf{Mac}(K', \epsilon\|A\|B\|r_A\|r_B)$ $m_A \leftarrow \epsilon\|\tau_A$		
	$\xrightarrow{m_A}$	
		if $(\mathsf{Vrf}(K', \epsilon\|A\|B\|r_A\|r_B, \tau_A) = \mathtt{false})$ abort **if** $(\epsilon = 1)$ upd_B kdf; upd_B $\tau'_B \leftarrow \mathsf{Mac}(K', r_B\|r_A)$
	$\xleftarrow{\tau'_B}$	
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		
$\tau'_A \leftarrow \mathsf{Mac}(K', r_A\|r_B)$		
	$\xrightarrow{\tau'_A}$	
		if $(\mathsf{Vrf}(K', r_A\|r_B, \tau'_A) = \mathtt{false})$ abort

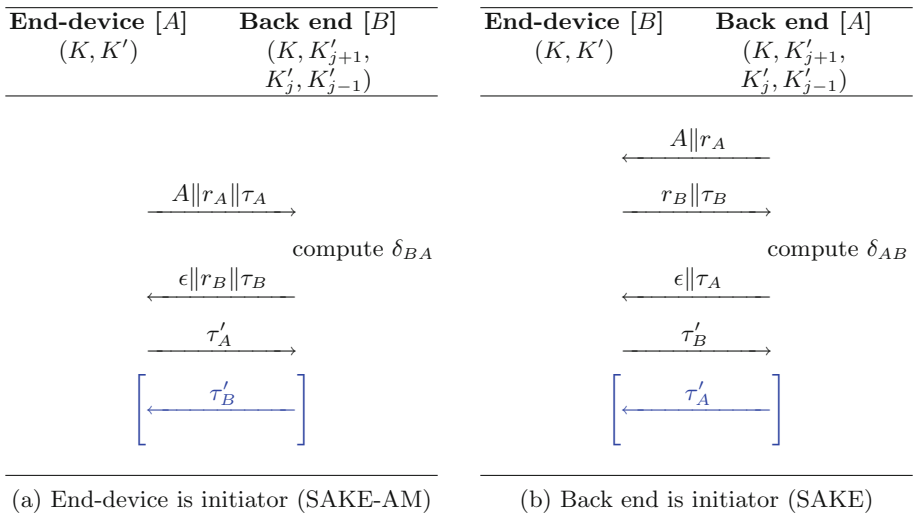
Fig. 2. SAKE protocol

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`.

3.3 SAKE-AM: A Complementary Mode of SAKE

From SAKE, we can derive an aggressive mode that allows any party to be either initiator or responder, and such that the smallest amount of calculation is always done by the *same* party.

In SAKE the initiator A owns the three keys K'_{j+1}, K'_j, K'_{j-1}, and the responder B does the lightest computations. In this mode B owns the three keys, and A does the smallest amount of calculation. The main idea is to skip the first SAKE message $A\|r_A$. Hence the roles between the two parties are swapped. This leads to other minor changes in message format compared to SAKE. Despite these differences, the messages and the calculations are essentially the same as in SAKE. This mode remains a sound and secure AKE protocol (according to Definition 5).[3] We call this mode *SAKE in aggressive mode* (SAKE-AM).



(a) End-device is initiator (SAKE-AM) (b) Back end is initiator (SAKE)

Fig. 3. Symmetric-key authenticated key exchange with forward secrecy between a low-resource end-device and a back-end server. Both parties may initiate the session. In some cases, the last message can be skipped.

This can be applied in the context of industrial IoT when a set of end-devices (e.g., sensors, actuators) communicate with a central server. When the end-device wants to initiate a communication, protocol SAKE-AM is launched.

[3] The proofs of soundness and security for SAKE-AM are essentially the same as for SAKE (see Sect. 5). They are given in the full version of the paper [8].

Otherwise (the server is initiator), SAKE is used (see Fig. 3). Therefore, the end-device always does the lightest computations.

4 A Random-Free Variant of SAKE

From SAKE, one can devise several variants. First, the three authentication keys K'_{j-1}, K'_j, K'_{j+1} can be replaced by two local counters c_A, c_B (respectively stored by A and B) that keep track of the evolution of the derivation master key K, with one static authentication master key K'.[4] On the initiator' side, the MAC verifications are then done with consecutive values of the counter $j-1$, j, $j+1$. Overall, the sequence of operations and the computations are similar to that of SAKE. This means mainly replacing function $x \mapsto \mathsf{Mac}(K'_j, x)$ with $x \mapsto \mathsf{Mac}(K', j\|x)$. Yet, this alternative implies the storage of two keys and one counter: K, K' and c_A/c_B, instead of two keys only: K and K'_{j-1}/K' (and, one the initiator' side only, one or two additional calls to update in order to compute K'_j and, possibly, K'_{j+1}).

Another, more interesting, variant is the following.[5] In SAKE, the pseudo-random values r_A, r_B are used to yield a fresh session key, and participate also in the authentication of the parties. Using new values during each session contributes to achieving these two tasks. Yet, these parameters are not the only ones to evolve throughout the successive protocol runs. The master keys do also. Therefore, one can consider removing the pseudo-random values from the messages. Without the pseudo-random values, several messages become cryptographically valid for each flow (instead of one only in SAKE). For instance, without r_A, party A may accept as second message either $\tau_B = \mathsf{Mac}(K'_j, B\|A)$, or $\tau_B = \mathsf{Mac}(K'_{j-1}, B\|A)$, or $\tau_B = \mathsf{Mac}(K'_{j+1}, B\|A)$. Likewise, without r_B, B may accept as third message either $0\|\tau_A$ or $1\|\tau_A$. Consequently, in this variant, we prefix each MAC-ed message with its index from 1 to 4 (but not the first one which carries only the initiator's identity).

The removal of the pseudo-random values enables a "mismatch attack". By "attack" we mean the following: an adversary is able to compel B to compute a message (message 4) which is *unaltered* by the adversary and expected by A, and yet A rejects this message as invalid. Although unpleasant, this "attack" does not break any claimed security property (in particular entity authentication). Moreover, this scenario cannot damage the synchronisation of the two parties. That is, if they start a new session, the latter completes successfully (if the adversary remains passive), as in SAKE.

In this variant, the length of the messages is shortened, and this avoids also calling the pseudo-random generation function. This is advantageous for low-resource devices. Nonetheless, the possibility provided by the aforementioned scenario is not what one usually expects from a security protocol. Consequently, for the practitioners for whom this mismatch attack is unacceptable, the SAKE protocol is more adequate.

[4] This alternative has been suggested by anonymous reviewers of Crypto 2019.

[5] We describe it from SAKE, but the same holds for SAKE-AM.

5 Security and Soundness for SAKE

In this section we prove that (i) SAKE is *sound*, and (ii) it is a *secure AKE* protocol according to Definition 5 given above.

5.1 Soundness of SAKE

We want to show that SAKE is *sound*, which essentially means that, once a correct session is complete, both parties have updated their respective internal state, are synchronised, and share the same (new) session key. We call a "benign" adversary an adversary that faithfully forwards all messages between an initiator A and a responder B.

Lemma 1. *Let A and B be respectively the initiator and the responder of a SAKE session. Let δ_{AB} be the gap between A and B with respect to the evolution of the master keys of both parties. The following conditions always hold:*

1. *$\delta_{AB} \in \{-1, 0, 1\}$, and*
2. *whatever the synchronisation state of A and B (i.e., whatever A and B are synchronised or not) when a new session starts, when that session completes in presence of a benign adversary, then*
   (a) A and B have updated their master keys at least once, and
   (b) A and B are synchronised (with respect to their master keys), and
   (c) A and B share the same session key.

In order to prove Lemma 1, we use the following notation. The messages exchanged during a session are numbered from 1 to 5. The notation "(i_A, i_B)" means that, when the session ends, the last valid message received by A is message of index i_A, and the last valid message received by B is message of index i_B. We call a (i_A, i_B)-*session* a session where the last message received by A is message i_A, and the last message received by B is message i_B. By convention $i_A = 0$ means that no message has been received by A.

It may happen that A send a first message which is not received by B. B cannot know if it has missed a first message. But this is of no consequence regarding the synchronisation between A and B (A may simply run the protocol anew). Therefore we do not use the value $i_B = 0$ (it is equivalent to $i_B = 5$). At initialisation (i.e., before the first run of the protocol), (i_A, i_B) is set to $(4, 5)$. Since A sends message $i \in \{3, 5\}$ only upon reception of a valid message $i - 1$, and B sends message $j \in \{2, 4\}$ only upon reception of a valid message $j - 1$, the only possible values for (i_A, i_B) are: $(0, 1)$, $(2, 1)$, $(2, 3)$, $(4, 3)$, and $(4, 5)$.

Proof. We prove Lemma 1. We first prove item 1.

Let c_A (resp. c_B) be a (virtual) monotonically increasing counter initialised to 0 that follows the evolution of the master keys held by A (resp. B). That is, c_A (resp. c_B) is increased each time the master keys K, K'_{j+1}, K'_j, K'_{j-1} (resp. K, K') are updated. The parameter δ_{AB} corresponds to the gap between A and B with respect to the evolution of their master keys, hence $\delta_{AB} = c_A - c_B$.

We prove item 1 by constructing iteratively Table 1b.

Before the first protocol run, A and B are synchronised. That is $\delta_{AB} = c_A - c_B = 0$, and $(c_A, c_B) = (i, i)$ (with $i = 0$). Therefore, A can validate τ_B (in message m_B) with the same key $K'_j = K'$ as B. Hence A computes $\delta_{AB} = 0$, and $\epsilon = 0$. Consequently, if one carries out the protocol run starting with $\delta_{AB} = 0$ and $\epsilon = 0$, for each possible value (i_A, i_B), one eventually gets the following:

- $(c_A, c_B) = (i, i)$ and $\delta_{AB} = 0$ after a $(0, 1)$-session,
- $(c_A, c_B) = (i + 1, i)$ and $\delta_{AB} = 1$ after a $(2, 1)$-session,
- $(c_A, c_B) = (i + 1, i + 1)$ and $\delta_{AB} = 0$ after a $(2, 3)$-session,
- $(c_A, c_B) = (i + 1, i + 1)$ and $\delta_{AB} = 0$ after a $(4, 3)$-session,
- $(c_A, c_B) = (i + 1, i + 1)$ and $\delta_{AB} = 0$ after a $(4, 5)$-session.

This corresponds to the first column of Tables 1a and b. As we can see, the only possible values for δ_{AB} after any session are 0 and 1. $\delta_{AB} = 0$ has already been investigated. Hence, starting with $\delta_{AB} = 1$ (i.e., $(c_A, c_B) = (i + 1, i)$), we look for all the values δ_{AB} may have when the session ends, considering any possible session.

$(c_A, c_B) = (i + 1, i)$ means that A is in advance with respect to B. In such a case, A succeeds in validating τ_B with K'_{j-1} (and, indeed, finds $\delta_{AB} = 1$). Then A uses $\delta_{AB} = 1$ and $\epsilon = 1$. If one carries out the protocol run using these two values, one gets three possible values for δ_{AB}: 1, 0, -1. This corresponds to the second column of Table 1b, and shows that a third value is possible for δ_{AB}, which is -1 (i.e., $(c_A, c_B) = (i, i + 1)$).

Then we restart the protocol with all possible sessions, assuming that $(c_A, c_B) = (i, i + 1)$ at the beginning of the run. This means that A is one step late with respect to B. In such a case, A succeeds in validating τ_B with key K'_{j+1} (and, indeed, finds $\delta_{AB} = -1$). Then A uses $\delta_{AB} = -1$ and $\epsilon = 0$. If one carries out the protocol run using these two values, we end with three possible values for δ_{AB} (third column of Table 1b): -1, 0 and 1, that have been explored already. This proves that, whatever the sequences of sessions, the only possible values for δ_{AB} are in $\{-1, 0, 1\}$.

Now we prove item 2 of Lemma 1.

We know that $\delta_{AB} \in \{-1, 0, 1\}$. For each possible value of δ_{AB} at the beginning of the session, the last line of Table 1b indicates the value of that parameter after a correct and complete session (i.e., a $(4, 5)$-session). As we can see, A and B are always synchronised (i.e., $\delta_{AB} = 0$) in such a case whatever the value of δ_{AB} when the session starts. Furthermore, the session key computation immediately precedes the last update of the derivation master key K. Hence, when a correct and complete session ends, A and B use the same derivation master key K to compute the session key. Therefore, using the same values r_A, r_B, A and B compute the same session key.

In addition, Table 1a shows that, whatever the synchronisation state of A and B (i.e., c_A and c_B) at the beginning of the session, after a correct and complete session, A and B have updated their internal state at least once (as the last line of the table, corresponding to a $(4, 5)$-session, indicates). □

Table 1. Possible values for δ_{AB} and (c_A, c_B) among all sequences of sessions in SAKE

(a) Possible values for (c_A, c_B)

session \ (c_A, c_B)	(i,i)	$(i+1,i)$	$(i,i+1)$
$(0,1)$	(i,i)	$(i+1,i)$	$(i,i+1)$
$(2,1)$	$(i+1,i)$	$(i+1,i)$	$(i+2,i+1)$
$(2,3)$	$(i+1,i+1)$	$(i+1,i+2)$	$(i+2,i+2)$
$(4,3)$	$(i+1,i+1)$	$(i+2,i+2)$	$(i+2,i+2)$
$(4,5)$	$(i+1,i+1)$	$(i+2,i+2)$	$(i+2,i+2)$

(b) Possible values for δ_{AB}

session \ δ_{AB}	0	1	-1
$(0,1)$	0	1	-1
$(2,1)$	1	1	1
$(2,3)$	0	-1	0
$(4,3)$	0	0	0
$(4,5)$	0	0	0

5.2 Security of SAKE

In order to prove that the protocol SAKE is a secure AKE protocol, we use the execution environment described in Sect. 2.1. We define the partnering between two instances with the notion of *matching conversations* (see Definition 8). 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 PRF, that is $\mathsf{update} : K \mapsto \mathsf{PRF}(K, x)$ for some (constant) value x.

Theorem 1. *The protocol SAKE is a secure AKE protocol, and for any probabilistic polynomial time adversary $\mathcal{A}$ in the AKE security experiment against protocol SAKE, we have*

$$\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE}(\mathcal{A}) \leq nq\left((nq-1)2^{-\lambda} + (q+1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}) + 2\mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Mac}}(\mathcal{C})\right)$$

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

where n is the number of parties, q the maximum number of instances (sessions) per party, λ the size of the pseudo-random values (r_A, r_B), and $\mathcal{B}$ is an adversary against the PRF-security of update*, $\mathcal{C}$ an adversary against the SUF-CMA-security of* Mac*, and $\mathcal{D}$ an adversary against the PRF-security of* KDF*.*

Proof. 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 $\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B})$ corresponding to the ability of an adversary $\mathcal{B}$ 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^t_j 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 [13,32], 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 in Game i.

Game 0. This game corresponds to the entity authentication security experiment described in Sect. 2.1. Therefore

$$\Pr[E_0] = \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE}(\mathcal{A})$$

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, 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 $\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B})$, each corresponding to the ability of an adversary $\mathcal{B}$ 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 $(q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B})$. Then we reduce the probability of the adversary $\mathcal{A}$ to win this game to the ability of an adversary $\mathcal{C}$ to forge a valid tag τ_B (resp. τ_A).

Therefore, we replace each function $\mathsf{update}(K') = \mathsf{PRF}(K', x)$ (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. Hence

$$\Pr[E_2] \leq \Pr[E_3] + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}) + \mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Mac}}(\mathcal{C})$$

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

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

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 get the indicated bound.

Now we prove the key indistinguishability security. Let E'_i be the event that an adversary win the key indistinguishability experiment 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 Sect. 2.1. Therefore

$$\Pr[E'_0] = \frac{1}{2} + \mathsf{adv}^{\mathsf{key\text{-}ind}}_{SAKE}(\mathcal{A}) = \frac{1}{2} + \mathsf{adv}_0$$

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

$$\mathsf{adv}_0 \leq \mathsf{adv}_1 + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE}(\mathcal{A})$$

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 $(q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B})$. Hence we replace each function $\mathsf{update}(K) = \mathsf{PRF}(K, x)$ (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 $\mathsf{G}^{\mathsf{update}}_0, \ldots, \mathsf{G}^{\mathsf{update}}_{q-2}$. 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

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

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

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

Collecting all the probabilities from Game 0 to 3, we get the indicated bound. □

6 Comparison with the DH Paradigm

The protocol SAKE provides a strong form of forward secrecy. Despite this result, it differs from a DH scheme in several ways beyond the intrinsic distinction between public-key and symmetric-key cryptography.

Concurrent Executions. Our protocol does not allow parallel executions. Indeed, since it is based on shared evolving symmetric keys, running multiple instances in parallel may cause some sessions to abort. A way to relax this restriction is that each party use separate master keys for concurrent executions. On the contrary, the DH scheme allows an (virtually) unlimited number of parallel executions.

KCI Attacks. The ephemeral DH scheme is resistant against KCI attacks, whereas our protocol is not (due to the dependency between the (updated) master keys). Moreover if an adversary succeeds in getting the key K' (or K'_j), it can compute the subsequent key (corresponding to K'_{j+1}). Hence the adversary can forge a message m_B in SAKE that brings the initiator to update its master keys twice consecutively. Therefore, that party is desynchronised with respect to an honest partner, with no possibility to resynchronise.

Note that KCI attacks affect also the static DH scheme (when a party uses a fixed DH share, whereas the other generates a fresh ephemeral one [25]).

Another consequence of the dependency of the master keys in SAKE, is that once the keys are compromised, an adversary can passively compromise all subsequent session keys. This is not the case in general with ephemeral DH. Yet, this is also true regarding non-DH public-key protocols (e.g., TLS-RSA), but also ephemeral DH (in some pathological cases) when reduced size (fixed) public parameters are used [5].

Computations. The DH scheme implies heavier computations (modular exponentiations, elliptic curve point multiplication) than SAKE which is solely built on symmetric-key functions. In practice, SAKE is likely more suitable to be implemented on constrained devices which have limited computational (and communication) capabilities.

7 Conclusion

We have described SAKE, an authenticated key exchange protocol in the symmetric-key setting. Although this protocol is solely based on symmetric-key algorithms, it provides perfect forward secrecy without requiring any additional procedure (e.g., resynchronisation phase) or functionality (e.g., shared clock). The underlying idea is to make the shared master keys evolve. We solve the synchronisation problem that stems from this evolving principle with an elegant and efficient solution.

SAKE guarantees that, whatever the synchronisation state of the involved parties prior to the session, both parties share a new session key, and their

internal state is updated and synchronised, once a correct session is complete: SAKE is self-synchronising. As in the public-key setting, our protocol allows an (virtually) unlimited number of sessions. Furthermore, we prove that SAKE is sound, and provide a formal proof of its security in a strong model.

Finally, we describe SAKE-AM, a complementary mode of our protocol, which, used in conjunction with SAKE, results in an implementation that gathers all the aforementioned properties (starting with forward secrecy). This implementation allows any party to be initiator or responder of a session, such that the smallest amount of calculation is always done by the same party. This is particularly convenient in the context of IoT where a set of (low-resource) end-devices communicates with a back-end server.

To the best of our knowledge, this is the first protocol with perfect forward secrecy in the symmetric-key setting that is comparable to the DH scheme, beyond the intrinsic distinction between public-key and symmetric-key cryptography.

Acknowledgment. We thank the anonymous reviewers for their valuable comments.

References

1. Signal. https://signal.org/
2. 3rd Generation Partnership Project: Technical Specifications 33. http://www.3gpp.org/DynaReport/33-series.htm
3. 3rd Generation Partnership Project: Technical Specifications 35. http://www.3gpp.org/DynaReport/35-series.htm
4. Abdalla, M., Bellare, M.: Increasing the lifetime of a key: a comparative analysis of the security of re-keying techniques. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 546–559. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_42
5. Adrian, D., et al.: Imperfect forward secrecy: how Diffie-Hellman fails in practice. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015, pp. 5–17. ACM Press, October 2015. https://doi.org/10.1145/2810103.2813707
6. Alwen, J., Coretti, S., Dodis, Y.: The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol. Cryptology ePrint Archive, Report 2018/1037 (2018). https://eprint.iacr.org/2018/1037
7. American National Standards Institute: ANSI X9.24-1:2009 Retail Financial Services Symmetric Key Management Part 1: Using Symmetric Techniques (2009)
8. Avoine, G., Canard, S., Ferreira, L.: Symmetric-key Authenticated Key Exchange (SAKE) with Perfect Forward Secrecy. Cryptology ePrint Archive, Report 2019/444 (2019). http://eprint.iacr.org/2019/444
9. 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, October 1997. https://doi.org/10.1109/SFCS.1997.646128
10. Bellare, M., Miner, S.K.: A forward-secure digital signature scheme. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 431–448. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_28
11. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. J. Cryptol. **21**(4), 469–491 (2008). https://doi.org/10.1007/s00145-008-9026-x

12. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_21
13. 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 (2006). https://doi.org/10.1007/11761679_25
14. Bellare, M., Yee, B.: Forward-security in private-key cryptography. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 1–18. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36563-X_1
15. Blake-Wilson, S., Johnson, D., Menezes, A.: Key agreement protocols and their security analysis. In: Darnell, M. (ed.) Cryptography and Coding 1997. LNCS, vol. 1355, pp. 30–45. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0024447
16. Boyd, C., Mathuria, A.: Protocols for Authentication and Key Establishment. Information Security and Cryptography. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-662-09527-0
17. Brier, E., Peyrin, T.: A forward-secure symmetric-key derivation protocol. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 250–267. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_15
18. Brzuska, C., Jacobsen, H., Stebila, D.: Safely exporting keys from secure channels. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 670–698. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_26
19. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the signal messaging protocol. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 451–466. IEEE, April 2017. https://doi.org/10.1109/EuroSP.2017.27
20. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Trans. Inf. Theory **22**(6), 644–654 (1976)
21. Diffie, W., van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. Des. Codes Crypt. **2**(2), 107–125 (1992)
22. Dousti, M.S., Jalili, R.: FORSAKES: a forward-secure authenticated key exchange protocol based on symmetric key-evolving schemes. Cryptology ePrint Archive, Report 2014/123 (2014). http://eprint.iacr.org/2014/123
23. GlobalPlatform: GlobalPlatform - Card Specification - Version 2.3.1, reference GPC_SPE_034, March 2018. https://www.globalplatform.org/specificationscard.asp
24. Günther, C.G.: An identity-based key-exchange protocol. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 29–37. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-46885-4_5
25. Hlauschek, C., Gruber, M., Fankhauser, F., Schanes, C.: Prying open Pandora's box: KCI attacks against TLS. In: Proceedings of the 9th USENIX Conference on Offensive Technologies, WOOT 2015, USENIX Association (2015)
26. International Organization for Standardization: ISO/IEC 11770–2 - Information technology - Security techniques - Key Management - Part 2: Mechanisms using Symmetric Techniques (2008)
27. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. Cryptology ePrint Archive, Report 2011/219 (2011). http://eprint.iacr.org/2011/219

28. Le, T.V., Burmester, M., de Medeiros, B.: Universally composable and forward-secure RFID authentication and authenticated key exchange. In: Bao, F., Miller, S. (eds.) ASIACCS 2007, pp. 242–252. ACM Press, March 2007
29. Park, T., Shin, K.G.: LiSP: a lightweight security protocol for wireless sensor networks. ACM Trans. Embed. Comput. Syst. **3**(3), 634–660 (2004)
30. Perrig, A., Szewczyk, R., Tygar, J., Wen, V., Culler, D.E.: SPINS: security protocols for sensor networks. Wireless Netw. **8**(5), 521–534 (2002)
31. Perrin, T., Marlinspike, M.: The Double Ratchet Algorithm (2016). https://signal.org/docs/specifications/doubleratchet/. Revision 1, 20/11/2016
32. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004). http://eprint.iacr.org/2004/332
33. Sornin, N., Luis, M., Eirich, T., Kramp, T.: LoRaWAN Specification, LoRa Alliance, version 1.0, July 2016
34. ZigBee Alliance: ZigBee specification. http://www.zigbee.org/download/standards-zigbee-specification/