Cryptanalysis of a "Strengthened" Key Exchange Protocol for IoT
(or When SAKE$^+$ turns out to be SAKE$^-$)

Loïc Ferreira

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

Abstract. In this paper we make an extensive analysis of SAKE$^+$ and SAKE$^+$-AM, two key exchange protocols. We show that several attacks are practicable against these protocols. This invalidates several claims made by the authors regarding the (security) properties of their protocols. Our results question also the correctness of the corresponding security proofs, made in the computational model (using the game-based methodology), and with the ProVerif verification tool.

Keywords: Authenticated key agreement · Cryptanalysis.

1 Introduction

In their seminal work on authenticated key exchange protocols [7], Bellare and Rogaway observe that the proliferation of incorrect and inefficient protocols is due to the lack of a meaningful definition of what such protocols aim at guaranteeing. Rather than designing a protocol by trial and error, they propose a formal framework in order to devise and analyse such kind of protocols. Following Bellare and Rogaway's work, different security models and methodologies have been proposed in order to analyse key exchange protocols intended to different settings: internet protocols [3, 9–11, 15–17, 22], mobile telephony [2, 21], RFID [4, 6, 24, 28] to name a few. These security models aim at capturing security properties adapted to the specific context the considered protocols aim at being deployed in. They constitute also useful tools to devise functional and secure key exchange protocols.

In [1], Aghili, Jolfaei, Abidin propose two authenticated key exchange protocols with forward secrecy dedicated to Internet of Things (IoT). These protocols build upon a previous work by Avoine, Canard, Ferreira [5]. In the latter, Avoine et al. propose two protocols (SAKE, SAKE-AM) solely based on the symmetric-key functions, and yet which provide forward secrecy. Aghili et al. propose a variant of these protocols, called SAKE$^+$ and SAKE$^+$-AM. They aim at improving Avoine et al.'s work by several aspects. First, SAKE$^+$ and SAKE$^+$-AM aim at keeping the same security properties as Avoine et al.'s protocols: entity authentication, key indistinguishability, and forward secrecy. In addition, SAKE$^+$ and SAKE$^+$-AM aim at being resistant to several attacks: "replay attacks", "time-based attack", and "tracking" (cf. [1], Sections 6.1 and 9). In support of these claims, security proofs made in the computational model (using the game-based methodology [8, 27]), and with the ProVerif verification tool [12] are also provided in [1].

1.1 Contributions

In this paper, we show that several claims made in [1] are not valid. Aghili et al. mention several (security) properties the protocols SAKE$^+$ and SAKE$^+$-AM aim at providing. We respect the same (informal) attack settings that are considered in [1] (in particular the powers allowed to the adversary), and we show that several of these properties can be broken. More specifically, we show that:

- the entity authentication property can be broken in SAKE$^+$ and SAKE$^+$-AM;
- a replay attack is feasible against SAKE$^+$-AM;
- time-based, and tracking attacks are doable against SAKE$^+$, and SAKE$^+$-AM;
- in addition, a disconnection attack is possible against SAKE$^+$, and SAKE$^+$-AM.

Table 1: List of attacks against the SAKE$^+$ and SAKE$^+$-AM protocols

	SAKE$^+$	SAKE$^+$-AM
Breakage of entity authentication (Section 4.1)	✓	✓
Replay attack (Section 4.2)	✗[1]	✓
Time-based attack (Section 4.3)	✓	✓
Tracking (Section 4.4)	✓	✓
Disconnection (Section 4.5)	✓	✓

1.2 Outline of the Paper

In Section 2 we describe the protocols SAKE$^+$ and SAKE$^+$-AM. In Section 3 we (informally) define the attacks that will be considered next. We explain how to apply these attacks against SAKE$^+$ and SAKE$^+$-AM in Section 4. Finally, we conclude in Section 5.

2 Description of the SAKE$^+$ and SAKE$^+$-AM Protocols

2.1 SAKE$^+$

Overview. The protocol SAKE$^+$ [1] is a two-party key exchange protocol based on a previous proposal called SAKE [5]. SAKE is an authenticated key exchange

[1] See Remark 7 in Section 4.2.

protocol solely built on symmetric-key functions, which provides forward secrecy. The latter property is achieved by using a key evolving scheme associated to a (re)synchronisation technique. Each party uses two types of keys: an authentication master key K', and a derivation master key K. The derivation master key K is used to compute session keys. K is updated with a one-way function in order to guarantee forward secrecy. Since desynchronisation may happen between the two parties A an B involved in a protocol run with respect to K, the authentication key K' is used to authenticate the parties but also to track the evolution of the internal state, and to resynchronise if necessary. In SAKE, the gap δ_{AB} between A and B is bounded ($\delta_{AB} \in \{-1, 0, 1\}$). That is, party A can only be in sync with B, or *one step* behind, or *one step* ahead. This allows both parties to resynchronise in the continuity of the protocol. Whatever the gap δ_{AB} between A and B when a new session starts, after a correct session, the two parties have updated their master keys, share the same session key, and are synchronised. In SAKE, the initiator A keeps the authentication master key corresponding to three consecutive epochs (K'_{j-1}, K'_j, K'_{j+1}) in order to be able to detect which epoch the responder belongs to (i.e., to compute δ_{AB} and to act accordingly). Yet, party A keeps only one value for the derivation master key K. The responder B keeps one pair of authentication and derivation master keys (K', K). If A and B are synchronised, $K'_j = K'$. If A is one step behind, $K'_{j+1} = K'$. If A is one step ahead, $K'_{j-1} = K'$.

Starting from SAKE, Aghili et al. make several changes and additions in order to achieve their goals. The master keys are updated in the same way as in SAKE using the one-way function update: $K \leftarrow \mathsf{update}(K)$, $K' \leftarrow \mathsf{update}(K')$. In addition, in SAKE$^+$, the responder B uses also evolving identities: $id_B \leftarrow \mathsf{update}(id_B \| K')$.

We describe below the protocol flow in SAKE$^+$ mainly based on the explanation given in [1], Section 6.2 and Figure 2 (see Figure 3).

First message. The initiator A sends $id_A \| r_A$ where id_A is the (fixed) identity of A, and r_A a pseudo-random value.

Second message. Upon reception of $A \| r_A$, B computes either $m_B = id_B \| r_B \| \tau_B$ (if $\phi = 0$) or $m_B = r_\alpha \| r_B \| \tau_B$ (if $\phi = 1$). The parameter id_B is the (ephemeral) identity of B, r_B and r_α are pseudo-random values, and $\tau_B = \mathsf{Mac}(K', id_B \| id_A \| r_B \| r_A)$ (in either case: $\phi = 0$ and $\phi = 1$).

The purpose of the ephemeral identity id_B is to forbid an adversary from being able to correlate several session related to B, and to track the latter. The flag ϕ indicates if id_B has been updated during the previous session. If $\phi = 0$, id_B has been renewed, hence it can be used to compute m_B (and then ϕ is set to 1). If $\phi = 1$, id_B is the same as in the previous session. In such a case, B computes $m_B = r_\alpha \| r_B \| \tau_B$.

Third message. As in SAKE, a party in SAKE$^+$ is able to establish at the same time a session with different party partners. Hence, party A may store different sets of parameters (in particular master keys and identity parameters), each corresponding to a different partner. When A receives m_B, it looks in its database.

If a match is found (case id_B), it retrieves the corresponding parameters: $(id_{B,j}, K'_j)$, $(id_{B,j-1}, K'_{j-1})$, $(id_{B,j+1}, K'_{j+1})$, K, r_{temp}.[2] For instance, if A and B are synchronised, then $id_{B,j} = id_B$ and $K'_j = K'$. The parameter $r_{temp} = (r'', r')$ is used to detect replay of previous messages m_B. The parameters r'' and r' correspond respectively to the two last values r_B received in a (valid) message m_B. To that point, if $r_B \in r_{temp}$, m_B is rejected. Otherwise, A uses the authentication master keys it has retrieved to verify the MAC tag τ_B.

Remark 1. Aghili et al. indicate that, when A is able to recognise the responder B with id_B (i.e., $\phi = 0$), it still tries the (at most) three authentication master keys K'_j, K'_{j-1}, and possibly K'_{j+1}, in order to verify the MAC tag τ_B. We observe that this is unnecessary if id_B closely follows the evolution of K' ($id_B \leftarrow \mathsf{update}(id_B \| K')$). Therefore, id_B can be used as (ephemeral) identity for B, but also as key identifier.

If A does not find a match in its database (case r_α), then for each existing entries, it tries all authentication keys until it finds the correct one. Then it retrieves the corresponding parameters $(id_{B,j}, K'_j)$, $(id_{B,j-1}, K'_{j-1})$, $(id_{B,j+1}, K'_{j+1})$, K, r_{temp}. If $r_B \in r_{temp}$, A aborts.

Eventually, if A did not abort so far, it updates r_{temp} with the new value r_B: first $r'' \leftarrow r'$, and then $r' \leftarrow r_B$.

If A is one step behind, it updates the parameters related to B a first time:

$$\begin{aligned}
&K && \leftarrow \mathsf{update}(K) \\
&id_{B,j-1} && \leftarrow id_{B,j} \\
&id_{B,j} && \leftarrow id_{B,j+1} \\
&id_{B,j+1} && \leftarrow \mathsf{update}(id_{B,j+1} \| K'_{j+1}) \\
&K'_{j-1} && \leftarrow K'_j \\
&K'_j && \leftarrow K'_{j+1} \\
&K'_{j+1} && \leftarrow \mathsf{update}(K'_{j+1})
\end{aligned}$$

Then it computes the session key: $sk \leftarrow \mathsf{KDF}(K, r_A, r_B)$, and it updates the parameters a second time. If A is in sync, it computes the session key, and updates the parameters once only. If A is one step ahead, the updates and the session key computation are delayed to a next step.

Finally, A computes the message $m_A = \epsilon \| \tau_A$. The parameter ϵ indicates if B is in sync ($\epsilon = 0$) or late ($\epsilon = 1$), and $\tau_A = \mathsf{Mac}(K'_i, id_{B,i} \| id_A \| r_A \| r_B)$. The parameter K'_i corresponds to the key used by A to verify correctly m_B, and $id_{B,i}$ is the associated identity parameter.

[2] The key K'_{j+1}, corresponding to the next epoch for A, is not always used during any session. Therefore, it can be computed from K'_j on the fly, and does not need to be stored. For the sake of explanation, we will make K'_{j+1} (and $id_{B,j+1}$) explicit in this paper.

Fourth message. When B receives m_A it verifies the MAC tag with K'. If it is not correct, B aborts. Then, if $\epsilon = 1$ (this indicates that B is late, and must catch up), B updates its parameters:

$$\begin{aligned} K &\leftarrow \mathsf{update}(K) \\ id_B &\leftarrow \mathsf{update}(id_B \| K') \\ K' &\leftarrow \mathsf{update}(K') \end{aligned}$$

Next, B computes the session key: $sk \leftarrow \mathsf{KDF}(K, r_A, r_B)$, updates its parameters (which means two updates if $\epsilon = 1$), and sets ϕ to 0.

Finally, B computes $m'_B = id_B \| \tau'_B$ with $\tau'_B = \mathsf{Mac}(K', r_B \| r_A)$.

Fifth message. If $\epsilon = 0$ (A is in sync now), A verifies m'_B with K'_j. If $\epsilon = 1$ (A was one step ahead at the beginning of the session, but now B must be ahead), A verifies m'_B with K'_{j+1}. If the MAC tag is not valid, A aborts. In addition, if $\epsilon = 1$, A computes the session key sk, and updates the master keys (but not the identity parameters, cf. [1], Section 6.2 and Figure 2). Finally, A computes $m'_A = id_{B,j} \| \tau'_A$ with $\tau'_A = \mathsf{Mac}(K'_j, r_A \| r_B)$.[3]

Party B verifies m'_A with K', and aborts if the result is `false`.

Remark 2. We observe that the message m'_A carries an identity parameter related to B. But no identity parameter is carried in message m_A. Therefore the usefulness of the identity parameter in these messages (or its absence) is questionable.

Notations. The following notations are used in Figure 3.

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 $id_{B,i}$ 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 value $id_{B,*}$ in m'_A is an identity parameter related to B. Yet, it is not explained in [1] how this value is chosen (e.g., in A's database, or from message m'_B).

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, and K' 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. $r' \leftarrow r_B$

[3] The message m'_A is computed as indicated at least when $\epsilon = 0$. That is, when A was not one step ahead at the beginning of the session. When A was one step ahead at the beginning of the session, it is not clear (at least to us) which identity parameter is used to compute m'_A (see also Section 4.5 and Section A).

The notations kdf, upd_A, upd'_A, and upd'_B are defined as follows:

– 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'_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. $K' \leftarrow \mathsf{update}(K')$

Moreover, $\mathsf{Mac}(k, m)$ denotes the MAC computation function that takes as input a secret key k, a message m, and outputs a tag τ. In turn, $\mathsf{Vrf}(k, m, \tau)$ 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`.

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 1.

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 entry is described with the pseudo-code given in Figure 2.

```
foreach $entry \in \mathsf{db}$
    if (verif-entry($entry, m_B$) = true)
        return $entry$
return $\emptyset$
```

Fig. 1: Pseudo-code of function find-entry

```
if ($\mathsf{Vrf}(K'_j, id_{B,j}\|id_A\|r_B\|r_A, \tau_B) = \mathtt{true}$)
    $\delta_{AB} \leftarrow 0$
    $K' \leftarrow K'_j$; kdf; $\mathsf{upd}'_A$; $\epsilon \leftarrow 0$
    return true
else if ($\mathsf{Vrf}(K'_{j-1}, id_{B,j-1}\|id_A\|r_B\|r_A, \tau_B) = \mathtt{true}$)
    $\delta_{AB} \leftarrow 1$
    $K' \leftarrow K'_{j-1}$; $\epsilon \leftarrow 1$
    return true
else if ($\mathsf{Vrf}(K'_{j+1}, id_{B,j+1}\|id_A\|r_B\|r_A, \tau_B) = \mathtt{true}$)
    $\delta_{AB} \leftarrow -1$
    $K' \leftarrow K'_{j+1}$; $\mathsf{upd}'_A$; kdf; $\mathsf{upd}'_A$; $\epsilon \leftarrow 0$
    return true
else
    return false
```

Fig. 2: Pseudo-code of function verif-entry

2.2 SAKE$^+$-AM

The protocol SAKE$^+$-AM is very close to SAKE$^+$. Compared to the latter, the first message ($A\|r_A$) is removed, and the roles (initiator and responder) are reversed. Moreover, what becomes then the first message is computed as $m_B = x\|r_B\|\tau_B$ with $x \in \{id_B, r_\alpha\}$ depending on ϕ, and $\tau_B = \mathsf{Mac}(K', id_B\|id_A\|r_B)$. Besides these differences, the calculations and messages are essentially the same as in SAKE$^+$.

2.3 Properties of SAKE$^+$ and SAKE$^+$-AM

Besides the properties that an authenticated key exchange protocol is supposed to guarantee (in particular entity authentication and key secrecy), SAKE$^+$ and SAKE$^+$-AM aim at providing the following properties.

Forward secrecy. This property aims at forbidding an adversary which corrupts a party from being able to compute a past session key. This property is ensured with two mechanisms. First, the key evolving scheme used to renew the master keys with the one-way function update (in particular the derivation master key K). Second, the (re)synchronisation mechanism which allows the two party A and B to be in sync after a correct session, whatever their gap at the beginning of the session.

Remark 3. We observe that in the security model presented by Aghili et al., each party is given "*its own unique master keys K, K'*" (cf. [1], Section 3). If each party uses unique master keys, either it can communicate with at most one party, which questions the practicality of such a setting, or the same master keys are shared by several parties, which trivially breaks forward secrecy.

```
A                                                  B
(Database db)                                      (K, K')

$r_A \xleftarrow{\$} \{0,1\}^\lambda$
                          $\xrightarrow{id_A \| r_A}$
                                                   $r_B \xleftarrow{\$} \{0,1\}^\lambda$
                                                   $\tau_B \leftarrow \mathsf{Mac}(K', id_B \| id_A \| r_B \| r_A)$

                                                   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

$\mathsf{upd}^r$
$\tau_A \leftarrow \mathsf{Mac}(K', \epsilon \| id_A \| id_{B,i} \| r_A \| r_B)$
$m_A \leftarrow \epsilon \| \tau_A$
                          $\xrightarrow{m_A}$
                                                   if $(\mathsf{Vrf}(K', \epsilon \| id_A \| id_B \| r_A \| r_B, \tau_A) = \mathtt{false})$
                                                       abort
                                                   if $(\epsilon = 1)$
                                                       $\mathsf{upd}'_B$

                                                   kdf; $\mathsf{upd}'_B$
                                                   $\phi \leftarrow 0$
                                                   $\tau'_B \leftarrow \mathsf{Mac}(K', r_B \| r_A)$
                                                   $m'_B \leftarrow id_B \| \tau'_B$
                          $\xleftarrow{m'_B}$
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; $\mathsf{upd}_A$

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

Fig. 3: The SAKE$^+$ and SAKE$^+$-AM protocols (based on our understanding of [1], Section 6.2 and Figure 2)

Resistance to tracking. The message m_B computed by the responder (resp. initiator) B in SAKE$^+$ (resp. SAKE$^+$-AM) carries B's ephemeral identity id_B or a pseudo-random parameter r_α (which replaces id_B if $\phi = 1$).[4] These parameters aim at forbidding an adversary from correlating several sessions corresponding to the same party, and tracking the latter.

Resistance to replay. In SAKE$^+$-AM, the parameter $r_{temp} = (r'', r')$, kept by the responder A, stores the two last values r_B received in a (valid) message m_B sent by the initiator B. The purpose is to forbid an adversary from replaying a message m_B (which contains r_B).

Remark 4. The same parameter r_{temp} is also used in SAKE$^+$ by the initiator A. We observe that A can detect anyway a replay of message m_B because the computation of its MAC tag involves the pseudo-random value r_A chosen by A (among other parameters). Therefore, r_{temp} seems useless in SAKE$^+$.

Concurrent executions. Concurrent executions means that two same parties are able to establish simultaneously multiple sessions. Since the protocols are based on *evolving* symmetric master keys, concurrent executions established with the same keys may cause some sessions to abort.

In order to allow concurrent executions, Aghili et al. apply the naive technique suggested in [5, 23]. That is, a party uses as many sets of master keys as parallel sessions it may be willing to establish. The drawback is that, for each party, the number of keys sets grows linearly with the number of concurrent sessions.

Remark 5. Using several sets of master keys contradicts also the uniqueness of the master keys presented in the security model. In addition, if a party owns many sets of master keys, Aghili et al. do not explain how its party partner may distinguish which set of master keys to use in order to process an incoming message. A solution is to attribute to each party as many (virtual) identities (and associated master keys) as concurrent sessions. Yet, this is not made clear in [1].

3 Description of the Attacks

In [1] Aghili et al. present several attacks that the protocols SAKE$^+$ and SAKE$^+$ aim at resisting to. These attacks, although not formally defined in [1], are informally described and backed with example scenarios. Aghili et al. refer also to security models proposed by other authors when considering some of the properties at stake. Below we recall these attacks.

[4] The parameter id_B is also included in other messages of the protocols.

3.1 Breakage of Entity Authentication

The security model used by Aghili et al. defines two main security properties: entity authentication and key indistinguishability (cf. [1], Section 3). They refer to the security model presented by Avoine et al. [5], which in turn is based on the security model proposed by Brzuska, Jacobsen, and Stebila [14]. We refer the reader to [1] for a detailed description of the security model.

Informally, entity authentication means that an instance π (i.e., an execution of the protocol handled by a given party A) is *partnered* with a unique instance π' (handled by the intended party partner B). Partnership is based on the notion of *matching conversations* [22]. This means that the instance π shares the same transcript of messages sent and received (but, possibly, the last message) as its instance partner π'.

Breaking the entity authentication property means that when an instance π "accepts" (i.e., deems that the protocol run is successful), there exists no other instance π' which is partnered with π, or there exist at least two other instances π' and π'' which are partnered with π.

3.2 Replay Attack

Although resistance to a replay attack is not formally defined in [1], a scenario illustrating what such an attack means is described (cf. [1], Section 5.3): "*the attacker first captures the valid message m_B related to the last session between the initiator and the responder A. Then, the adversary resends the captured message m_B to A, repeatedly.* [...] *After the verification of* [m_B] *A will compute* [m_A.] *Now, A sends the created message to the initiator B. Hence, the adversary succeeded in forcing the party A to perform unnecessary calculations.*"

In this scenario, the adversary eavesdrops on a valid message sent by one party B (the initiator in the described scenario) to its party partner A. The adversary replays this message to A. Then the responder A accepts the message as valid, computes a valid response, and sends it. The victim A does the same as long as the adversary replays this same message. Therefore such a replay attack may be informally defined as the ability for an adversary to compel a legitimate party to do useless computations and sendings of messages (useless in the sense that these operations do not lead to the completion of a session with a legitimate party, even if the latter receives the messages sent by the victim upon the adversary's action).

3.3 Time-based Attack

Aghili et al. describe a time-based scenario where an adversary uses the time spent in calculations by some party A upon reception of a message sent by a party B, in order to recognise the latter (cf. [1], Section 5.2): "*Timeful attack is an attack where an adversary identifies which responder has just been authenticated by an initiator by observing the amount of time required to authenticate the responder* [...]. *This attack can be performed against an AKE protocol in which* *the initiator must perform exhaustive search to find the responder's identity. The adversary then exploits the fact that it takes the same amount of time for the initiator to authenticate and accordingly respond to a particular responder in every execution of the protocol. This allows the adversary to detect which responder has been authenticated by the initiator.*"

We stress that this scenario allows *recognising* a legitimate party, not necessarily *identifying* it. More precisely, in this scenario, party A tries all keys stored in a database until it finds the one corresponding to B (i.e., the key which correctly verifies the message sent by B). Therefore, the adversary can use the time spent by A as an index to recognise B in subsequent sessions, and to discriminate B from other legitimate parties.

In addition, Aghili et al. refer to the security model defined by Avoine, Coisel, and Martin [6], which captures active adversaries. We refer the reader to [6] for a detailed description of the security model.

3.4 Tracking

Aghili et al. presents the tracking attack as the ability for an adversary to correlate two different sessions to the same party (cf. [1], Section 5.4): "*an adversary can easily relate all the messages m_B that has been captured from valid sessions between B and A. This is because the initiator B attaches its fixed identity, B, to m_B in plaintext; hence, the adversary is able to eavesdrop this identity and track B.*"

The above scenario involves a passive adversary. But another one (cf. [1], Section 6.1) which illustrates a different tracking attack involves an active adversary (it sends messages to the victim): "*An adversary creates a message $A \| r_A$, sends it continually to B, and receives a response containing the responder's identity (B) for each message. By linking the identities in the responses, the adversary can successfully trace the target responder.*"

In addition, Aghili et al. refers to the security model defined by Ouafi and Phan [25] when describing the tracking attack. This model captures active adversaries. We refer the reader to [25] for a detailed description of the security model.

3.5 Disconnection

In addition to the different attacks mentioned in [1], we introduce yet another attack that we call "disconnection". The purpose of this scenario is to forbid everlastingly two parties A and B from being able to successfully complete a session. Of course, such a scenario is always possible if the adversary is constantly active, and forbids one of the two parties from receiving one of the messages needed to complete the protocol run. Therefore, by disconnection attack, we mean that the adversary does not need to be constantly active. It intervenes at most a few times: it can only forward, alter, and drop any message exchanged between honest parties, then it stops being active. Afterwards, the two targeted parties are not able to communicate together anymore. That is, they become "disconnected" forever.

4 Attacks against SAKE$^+$ and SAKE$^+$-AM

In this section, we describe attacks against SAKE$^+$ and SAKE$^+$-AM that are in compliance with the scenarios mentioned by Aghili et al. (see Section 3), and the (formal or informal) security experiments and adversarial models they consider in [1].

4.1 Breakage of Entity Authentication

As recalled in Section 3.1, the model used by Aghili et al. to prove the security of SAKE$^+$ and SAKE$^+$-AM incorporates the entity authentication property. In addition, Aghili et al. claim that these protocols inherit from the SAKE and SAKE-AM protocols [5], and, as such, do guarantee the same security properties (cf. [1], Sections 1 and 6.1). This includes entity authentication. We show below that this claim is invalid, and that the entity authentication property can be broken in SAKE$^+$ and SAKE$^+$-AM.[5]

Scenario. In SAKE$^+$, upon reception of an initial message $id_A \| r_A$ from the initiator A, the responder B computes a message m_B as $id_B \| r_B \| \tau_B$ (if $\phi = 0$) or $r_\alpha \| r_B \| \tau_B$ (if $\phi = 1$). The parameter id_B is the (ephemeral) identity of B, r_α and r_B are pseudo-random values. In either case ($\phi = 0$ and $\phi = 1$), $\tau_B = \mathsf{Mac}(K', id_B \| id_A \| r_B \| r_A)$. That is, r_α is not included in the computation of the MAC tag τ_B.

Upon reception of m_B, A retrieves from its database the correct authentication key in order to verify τ_B. If $m_B = id_B \| r_B \| \tau_B$, A uses id_B to find the key, otherwise ($m_B = r_\alpha \| r_B \| \tau_B$) no match is found with any id_B value stored in the database, and A exhaustively tries all authentication keys existing in the database.

Since r_α is not included in the computation of the MAC tag τ_B, an adversary can alter id_B (if $\phi = 0$) and r_α (if $\phi = 1$) without A being able to notice the modification. Yet, A is able to find the authentication master key which verifies correctly the MAC tag (since the latter is valid). Therefore, A can send a valid response, and eventually the session is successful. In such a case, A and B do not share the same transcript of messages sent and received. That is, the instance executed by A (resp. B) has no partner.

The same holds for SAKE$^+$-AM. The initial message sent by B to A is either $m_B = id_B \| r_B \| \tau_B$ (if $\phi = 0$) or $m_B = r_\alpha \| r_B \| \tau_B$ (if $\phi = 1$). In either case, $\tau_B = \mathsf{Mac}(K', id_B \| id_A \| r_B)$. The message m_B is processed by A in the same way as in SAKE$^+$.

[5] But not in SAKE and SAKE-AM.

Since the instances executed by the two parties A and B "accept" without having a partner, this scenario breaks the entity authentication property in SAKE$^+$ and SAKE$^+$-AM.

We will see in Sections 4.2 and 4.3 that this flaw results in other consequences.

Remark 6. In SAKE$^+$ and SAKE$^+$-AM, the two last messages are $m'_B = id_B \| \tau'_B$ with $\tau'_B = \mathsf{Mac}(K', r_B \| r_A)$, and $m'_A = id_{B,*} \| \tau'_A$ with $\tau'_A = \mathsf{Mac}(K'_j, r_A \| r_B)$. Since id_B (resp. $id_{B,*}$) is not included in the computation of τ'_B (resp. τ'_A), the adversary can also alter this parameter. Nonetheless, it is not clear (at least to us) how the identity parameters id_B and $id_{B,*}$ are used by A and B to process the messages m'_B and m'_A. That is, it is not clear if the session is successful or aborts, when this identity parameter is modified.[6]

Mitigation. In order for A to be able to detect that m_B is modified, r_α must be included in the computation of the MAC tag τ_B. That is, if $\phi = 1$, $\tau_B = \mathsf{Mac}(K', r_\alpha \| id_A \| r_B \| r_A)$ (or $\tau_B = \mathsf{Mac}(K', r_\alpha \| id_B \| id_A \| r_B \| r_A)$) in SAKE$^+$, and $\tau_B = \mathsf{Mac}(K', r_\alpha \| id_A \| r_B)$ (or $\tau_B = \mathsf{Mac}(K', r_\alpha \| id_B \| id_A \| r_B)$) in SAKE$^+$-AM.

In addition, the identity parameter must be involved in the computation of the two MAC tags of the messages m'_A and m'_B, in SAKE$^+$ and SAKE$^+$-AM.

4.2 Replay Attack

In [1], Sections 1.1, 5.3, 6.3 and 9, Aghili et al. claim that SAKE$^+$-AM resists to the same kind of scenario as described in Section 3.2. Below, we show that this claim is invalid.

Scenario. In order to thwart a replay attack, the responder A in SAKE$^+$-AM stores the two last pseudo-random values r_B sent by the initiator B (they are stored in the list $r_{temp} = (r''_B, r'_B)$). If the same initial message is received once again by A, such a replay is supposed to be detected by comparing the received value r_B with the values stored in r_{temp}. This countermeasure can be bypassed if the adversary uses three different initial messages corresponding to the (same) previous epoch. That is, three initial messages computed under the same authentication key $K' = K'_{j-1}$ corresponding to the previous epoch with respect to the receiver A.

The adversary does the following: it eavesdrops three times consecutively an initial message m_B sent by the initiator B, and not received by A (dropped by the adversary). The first message is computed by B in state $\phi = 0$ or $\phi = 1$ with some key K', and the second and third messages are computed in state $\phi = 1$ with the same key K'. These three initial messages are computed with the same authentication key because B has not received any response from A. Hence, it has not updated its master keys.

Then, B sends once again a new initial message (computed in state $\phi = 1$).

[6] In this regard, see also Section 4.5 and Section A.

The adversary does not intervene, and this message is received by A. Upon reception of this (fourth) initial message, A acts accordingly with the protocol, and the session is eventually completed by the two parties. To that point, the three messages obtained by the adversary belong to the previous epoch on A's perspective (because A has updated its master keys once).

With high probability the pseudo-random value r_B carried in the three messages eavesdropped by the adversary are pairwise distinct, and are all different from any element in r_{temp} (which includes now the last r_B value sent by B in the fourth initial message). Therefore, the adversary can alternately send to A its three messages. Since they are computed with a valid authentication key (K'_{j-1}), the MAC tag is correctly verified. Since the three r_B values are pairwise distinct, and do not belong to r_{temp}, they are not rejected as replayed messages. Therefore, A makes all the corresponding computations, and eventually sends a valid response (message m_A) as long as the adversary keeps sending the three messages alternately. Moreover, since at least two among the three messages used by the adversary are computed by B under the state $\phi = 1$, A makes the maximum amount of computations when it receives these messages. Indeed, in such a case B's (ephemeral) identity id_B is replaced in message m_B with a pseudo-random value r_α. Hence, upon reception of m_B, A must try (at most) the three possible authentication keys K'_j, K'_{j-1}, K'_{j+1}, for each entry stored in its database until a correct key is found.[7]

If the adversary wants to be sure that the three initial messages correspond to $\phi = 1$ in order to compel A to make as most computations as possible, it can eavesdrop and block the fourth message sent by B, and use the second, third, and fourth initial messages. Or the adversary can merely replace the first parameter (id_B or r_α) in the first eavesdropped message m_B with a pseudo-random value r^*_α. As explained in Section 4.1, this change remains undetected by A.

The alternate use of the three initial messages allows the adversary to "flush" the list r_{temp} on A's side. More generally, if A stores $n \geq 1$ previous values r_B, the same scenario can be done with $n + 1$ initial messages m_B (each of them corresponding to the (same) previous epoch).

If B does never send a third or a fourth initial message[8] m_B, then this scenario is an easy way to "kill" B once and for all. That is, B becomes unavailable forever.

This scenario is interrupted when the two legitimate parties B and A succeed in completing a new session. Then, the master keys are updated (in particular the authentication master key corresponding to the previous epoch), and this obsoletes the messages m_B *currently* used by the adversary. Yet, the latter can reload its "ammunition".

Remark 7. In SAKE$^+$, the first message corresponds to $A \| r_A$ where r_A is a pseudo-random value. If an adversary replays (or compute) such a message, the responder replies with a valid message.

[7] If K'_{j+1} is not stored in the database but computed on the fly, this adds a supplementary computation for each tested entry not corresponding to B.

[8] Or a fifth initial message.

Although such a scenario complies with Aghili et al.'s description of a replay attack, they do not seem to consider it as a valid attack. We do not either. Indeed such an initial message *must* be accepted by the receiver. Likewise, in many other protocols, the responder must reply to an initial message (e.g., TLS 1.2 [18]). But sending such an initial message is not enough for the responder to "accept", and for the session to end successfully. In SAKE-AM [5] also the initial message can be replayed if it belongs to the previous epoch, but this is not sufficient to authenticate the initiator, and eventually the session aborts.

In contrast, in TLS 1.3 with 0-RTT mode [26] the server must deem the initial message (Client Hello) as authentic, and execute the request herein included [20]. Consequently, mitigations are necessary (cf. [26], Section 8).

Mitigation. The scenario described above is possible because, in SAKE$^+$-AM, only the last two received values r_B are stored by A.

In order to thwart such a scenario, A can keep track of all previously received values. For instance, this can be done with a Bloom filter [13] or a Cuckoo filter [19]. 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. Besides storing an additional parameter, such a technique implies implementing and executing hash functions. Party A may store one such filter per party it may communicate with, or one global filter for all possible party partners.

4.3 Time-based Attack

In [1], Sections 1, 5.2 and 9, Aghili et al. claim that the SAKE$^+$ protocols resist to the same kind of scenario as described in Section 3.3. We show below that this statement is not valid. We also show that this scenario applies to SAKE$^+$-AM.

Scenario. In SAKE$^+$, upon reception of an initial message $id_A \| r_A$ from the initiator A, the responder B sends either $m_B = id_B \| r_B \| \tau_B$ (if $\phi = 0$) or $m_B = r_\alpha \| r_B \| \tau_B$ (if $\phi = 1$). The parameter id_B is the (ephemeral) identity used by B, r_α and r_B are pseudo-random values, and $\tau_B = \mathsf{Mac}(K', id_B \| id_A \| r_B \| r_A)$ (whatever the value of ϕ). The value ϕ depends on what happened during the previous session. If, during the previous session, B has received a valid third message (m_A), then id_B and K' are updated, and ϕ is set to 0. Otherwise, id_B and K' remain the same, and $\phi = 1$. Therefore, ϕ indicates if id_B has been changed or not. If not ($\phi = 1$), then B replaces id_B (which remains equal to the ephemeral identity used during the previous session) with a pseudo-random value r_α. The purpose is to forbid an adversary from correlating the previous and current sessions (the same value id_B is not used twice consecutively).

When m_B carries id_B, A can look for this value in its database, and retrieve the corresponding derivation and authentication master keys. If m_B carries r_α, with high probability, A does not find a match in its database. Therefore the only way in order to process the message m_B is to try, for all existing entries in the database, the different authentication keys, and check if one key correctly verifies the MAC tag τ_B. Hence, when $m_B = r_\alpha \| r_B \| \tau_B$, the time spent by A to find the correct authentication key allows an adversary to recognise which party B communicates with A (the time measurement done by the adversary is used as an index that designates to B).

The same holds with respect to SAKE$^+$-AM. In this case, the message m_B is the initial message sent by the initiator B to the responder A. This message is equal to $id_B \| r_B \| \tau_B$ (if $\phi = 0$) or $r_\alpha \| r_B \| \tau_B$ (if $\phi = 1$), with $\tau_B = \mathsf{Mac}(K', id_B \| id_A \| r_B)$ whatever the value of ϕ. Upon reception of m_B by A, the message is processed in the same way as in SAKE$^+$.

This can be used in *any* case (i.e., whatever the value of ϕ) by an adversary in order to recognise a responder (resp. an initiator) B in SAKE$^+$ (resp. SAKE$^+$-AM).

First flavour. In SAKE$^+$, the adversary poses as a person-in-the-middle between A and B. When $m_B = id_B \| r_B \| \tau_B$ (i.e., $\phi = 0$), the adversary replaces id_B with a pseudo-random value r^*_α, and sends $m^*_B = r^*_\alpha \| r_B \| \tau_B$ to A. As explained in Section 4.1, A is not able to detect the change because the computation of the MAC tag τ_B does not involve r_α. With high probability, A does not find a match to r^*_α in its database. Therefore, it tries all existing entries until it finds the correct one. More precisely, for each entry, A retrieves $(id_{B,j}, K'_j, id_{B,j-1}, K'_{j-1}, id_{B,j+1}, K'_{j+1})$, and verifies the MAC tag τ_B with $id_{B,i}$ and K'_i, $i \in \{j-1, j, j+1\}$. Hence, A eventually finds an entry such that a pair $(id_{B,i}, K'_i)$, $i \in \{j-1, j, j+1\}$, allows verifying correctly τ_B. Indeed A updates $id_{B,j}$ and K'_j the same way as B does. Therefore, whatever if A is in sync with B, or one step behind (but not necessarily if A is one step ahead[9]), it will find a consistent pair $(id_{B,i}, K'_i)$ that allows verifying the MAC tag τ_B computed by B.[10]

The value id_B is output by the update function, and r_α is a λ-bit pseudo-random binary string. Although it is not explicitly stated in [1], we may assume that both values are indistinguishable (at least their size is the same) since they correspond to the same parameter in m_B. Therefore, even when $m_B = r_\alpha \| r_B \| \tau_B$ (i.e., if $\phi = 1$), the adversary has to replace r_α with a pseudo-random λ-bit value r^*_α. If the adversary is able to distinguish either case, then it does not need to intervene when $\phi = 1$.

The same holds with respect to SAKE$^+$-AM. The adversary acts as a person-in-the-middle between B and A. It replaces id_B in $m_B = id_B \| r_B \| \tau_B$ (if $\phi = 0$) or r_α in $m_B = r_\alpha \| r_B \| \tau_B$ (if $\phi = 1$), with a pseudo-random value r^*_α, and sends $m^*_B = r^*_\alpha \| r_B \| \tau_B$ to A. The message m^*_B is processed by A in a similar way as in SAKE$^+$.

Second flavour. The specific scenario presented above is possible (against SAKE$^+$ and SAKE$^+$-AM) because the adversary can alter m_B without A being able to

[9] See Section 4.5.

[10] We recall that, when A is in sync or one step behind, it updates $id_{B,j}$, K'_j as follows: $id_{B,j} \leftarrow \mathsf{update}(id_{B,j} \| K'_j)$, $K'_j \leftarrow \mathsf{update}(K'_j)$.

notice the change. However, even if A is able to verify if m_B has been modified, the same kind of attack can be done.

Indeed, in SAKE$^+$, A does an exhaustive search in its database when it receives a genuine message $m_B = r_\alpha \| r_B \| \tau_B$ (i.e., when $\phi = 1$) – not modified by the adversary. The adversary can trigger the computation of such a message. It merely waits for B to send m_B in response to A's initial message, and forbids A from receiving m_B. Then ϕ is set to 1. Very likely, A sends anew an initial message, which B responds to with $m_B = r_\alpha \| r_B \| \tau_B$. Next, A and B proceed according to the protocol, and, eventually, the session is successfully completed.

Likewise, in SAKE$^+$-AM, if the adversary forbids A from receiving an initial message m_B sent by B, ϕ is set to 1. Then, very likely, B sends a new initial message of the form $m_B = r_\alpha \| r_B \| \tau_B$.

Time estimate. In SAKE$^+$ and SAKE$^+$-AM, when A tries all authentication keys existing in its database in order to verify the MAC tag τ_B, it makes three MAC computations. Despite the fact that three computations are done for each entry, it may happen that the time spent by A to try two entries that are close (e.g., consecutive) in its database be similar. Hence, the adversary may not be able to discriminate the two corresponding parties [6]. Indeed, being able to recognise any party depends on how tight is the time measurement made by the adversary. We observe that against SAKE$^+$-AM, the adversary can still be winner.

Indeed, in SAKE$^+$-AM, the adversary can replay messages m_B (see Section 4.2), and replace the first parameter of the message (id_B or r_α) with a pseudo-random value r_α^* without the receiver A being able to detect the change (see Section 4.1). Therefore, based on these multiple measurements, the adversary can get a more accurate estimate of the time spent by A to process m_B, and eventually discriminate two such "close" parties.

Mitigation. A first mitigation in SAKE$^+$ consists in including r_α in the computation of the MAC tag τ_B.[11] That is, when $\phi = 1$, B computes $\tau_B = \mathsf{Mac}(K', r_\alpha \| id_A \| r_B \| r_A)$ (or $\tau_B = \mathsf{Mac}(K', r_\alpha \| id_B \| id_A \| r_B \| r_A)$). Upon reception of m_B, if A does not find a match to r_α in its database, it recovers the correct entry by comparing $\mathsf{Mac}(K'_i, r_\alpha \| id_A \| r_B \| r_A)$ (or $\mathsf{Mac}(K'_i, r_\alpha \| id_{B,i} \| id_A \| r_B \| r_A)$) with τ_B for all authentication keys K'_i existing in its database (and the corresponding identity parameter $id_{B,i}$).

The same holds for SAKE$^+$-AM. If $\phi = 1$, B computes $\tau_B = \mathsf{Mac}(K', r_\alpha \| id_A \| r_B)$ (or $\tau_B = \mathsf{Mac}(K', r_\alpha \| id_B \| id_A \| r_B)$), and A verifies the MAC tag accordingly.

As an additional mitigation for SAKE$^+$ and SAKE$^+$-AM, A can try to equalise the time spent to process the message m_B. That is, A tries always all authentication keys it owns, even when it finds the correct one. Therefore, the time spent by A to process m_B corresponds always to the time needed to explore the whole database. The drawback is that it compels A to make the maximum amount of computations anytime.

[11] This is also used to mitigate the attack described in Section 4.1.

Another possibility is to search the database randomly each time [6]. One option is to randomly choose the starting index. Another option is for A to randomly generate a permutation to reorder the indexes of the database, and explores the database accordingly. Party A stops when the correct entry is found.[12] In either case, the time spent to find the correct key (i.e., B's identity) is no more related to its "real" index in the database.

4.4 Tracking

In [1], Sections 1.1, 6.1 and 9, Aghili et al. claim that SAKE$^+$ and SAKE$^+$-AM resist to the same kind of scenario as described in Section 3.4. We show below that this claim is not true.

Scenario. The adversary applies the same technique as the one used in the time-based attack: in SAKE$^+$ (resp. SAKE$^+$-AM), it compels the initiator (resp. responder) A to try all authentication keys for each entry existing in its database until it finds the correct one that allows verifying the message m_B sent by the responder (resp. initiator) B. The time spent by A to explore the database is used by the adversary as an index to recognise which party B, A is communicating with. The adversary can do this for each session in order to detect which a given party B is involved in. Hence the adversary can track that party.

Mitigation. The tracking attack is feasible because the adversary is able to compel A to explore its database. Hence the mitigations are the same as those proposed for the time-based attack (see Section 4.3).

4.5 Disconnection

In this section, we show that it is possible in SAKE$^+$ and SAKE$^+$-AM to forbid everlastingly two parties A and B from being able to successfully complete a session, without an adversary needing to be constantly active. The adversary intervenes once only (to drop one message), or possibly never (if errors occurs on the communication channel).

Scenario. The adversary relies almost only on the peculiarities of SAKE$^+$ in order to achieve its goal.[13]

[12] We observe that A stops the search when the correct *entry* is found. But still, it may be necessary that A try all authentication keys corresponding to that entry. This aims at precluding possible attacks based on the fact that for one party B, the first tried key is the correct one, whereas for another party C communicating with A, the second or third key is the correct one. Besides, if A does not store K'_{j+1} but computes the key only when necessary, this additional calculation may favour the adversary since it increases the time discrepancy.

[13] See Section A for a detailed description of the scenario.

The SAKE$^+$ protocol uses the same technique as the SAKE protocol in order to update the master keys, and to maintain the synchronisation between A and B [5]. That is, whatever the gap between A and B at the beginning of a session, once a correct session ends, both parties have updated their keys, and are synchronised. Moreover, in SAKE$^+$, the computation of the identity parameter id_B depends on the authentication master key: $id_B \leftarrow \mathsf{update}(id_B \| K')$.

In a correct session, both the master keys and the identity parameter are updated together on the responder's side (B). However on the initiator's side (A), the master keys and the identity parameters are updated only when A is either in sync with B or one step behind. When A is one step ahead, it updates only its master keys, but not the identity parameters.[14] Therefore, if A is one step ahead, and A and B start a new session, once the session ends correctly, B has updated its master keys and the identity parameter, but A has updated the master keys only. Hence A and B are synchronised with respect to the master keys, but desynchronised in regard to the identity parameter.

More precisely, let us assume that the current epoch with respect to A is t, and B still belongs to epoch $t-1$. That is B stores

$$id_B = id_{B,t-1}, K' = K'_{t-1}, K = K_{t-1}$$

and A stores

$$\begin{gathered}(id_{B,j}, K'_j, K_j),\, (id_{B,j-1}, K'_{j-1}, K_{j-1}),\, (id_{B,j+1}, K'_{j+1}, K_{j+1}) \\ = \\ (id_{B,t}, K'_t, K_t),\; (id_{B,t-1}, K'_{t-1}, K_{t-1}),\; (id_{B,t+1}, K'_{t+1}, K_{t+1})\end{gathered}$$

Then A and B complete successfully a session. Party B has updated its master keys and identity parameter (twice because it must resynchronise). That is, B computes first

$$\begin{array}{lll} id_B & \leftarrow \mathsf{update}(id_B \| K') & = id_{B,t} \\ K' & \leftarrow \mathsf{update}(K') & = K'_t \\ K & \leftarrow \mathsf{update}(K') & = K_t \end{array}$$

and then

$$\begin{array}{lll} id_B & \leftarrow \mathsf{update}(id_B \| K') & = id_{B,t+1} \\ K' & \leftarrow \mathsf{update}(K') & = K'_{t+1} \\ K & \leftarrow \mathsf{update}(K') & = K_{t+1} \end{array}$$

In turn, the entry stored by A and related to B is changed from

$$(id_{B,t}, K'_t, K_t), (id_{B,t-1}, K'_{t-1}, K_{t-1}), (id_{B,t+1}, K'_{t+1}, K_{t+1})$$

into

$$(id_{B,t}, K'_{t+1}, K_{t+1}), (id_{B,t-1}, K'_t, K_t), (id_{B,t+1}, K'_{t+2}, K_{t+2})$$

[14] To be fair, this resembles to an oversight from the authors of [1]. Yet, this is how it is described in [1], Sections 6.1 and 6.2, and Figure 2. We can do nothing but to stick to the explanation of the protocol as it is given.

because A does not update the identity parameters.

To that point, when A updates the identity parameters corresponding to B (during a subsequent session), it gets $\tilde{id}_{B,i} = \mathsf{update}(id_{B,i-1} \| K'_i)$, $i \in \{t, t+1, t+2\}$. On B's side, the evolution of id_B closely follows that of K': $id_B \leftarrow \mathsf{update}(id_B \| K')$. Therefore, with high probability, $id_B \neq \tilde{id}_{B,i}$, $i \in \{t, t{+}1, t{+}2\}$.

When A starts a new session, the identity parameter sent by B in $m_B = id_B \| r_B \| \tau_B$ (with $\tau_B = \mathsf{Mac}(K', id_B \| id_A \| r_B \| r_A)$) has no match in A's database. Therefore, A tries the different authentication master keys, and the corresponding identity parameter found in its database. Since $id_B \neq \tilde{id}_{B,i}$, $i \in \{t, t{+}1, t{+}2\}$, the verification of the MAC tag τ_B yields always `false`. The same recurs in the next sessions. Hence A and B are unable to communicate anymore.

In SAKE$^+$-AM, the responder (resp. initiator) updates the master keys and the identity parameters in the same way as the initiator (resp. responder) in SAKE$^+$. The same sequence of operations as in SAKE$^+$ is executed. Therefore, the same scenario can be applied against SAKE$^+$-AM.

Making party A one step ahead. Party A is one step ahead with respect to B if A and B are first synchronised, and B does not receive message m_A. This may happen either if the adversary forbids B from receiving m_A (the adversary intervenes once), or if the message is not received or is altered because of errors in the communication channel (the adversary does not intervene at all).

Mitigation. When the initiator (resp. responder) A in SAKE$^+$ (resp. SAKE$^+$-AM) receives the fourth (resp. third) message of the protocol flow, and $\epsilon = 1$, A must update the identity parameters when it updates the master keys also.

5 Conclusion

In this paper we have made an extensive analysis of SAKE$^+$ and SAKE$^+$-AM, two key exchange protocols. We have shown that several claims made in [1] regarding the (security) properties of these protocols are not valid. More specifically, we have described an attack against mutual authentication, as well as tracking, replay, and time-based attacks. We have also presented an attack which forbids two parties from being able to communicate anymore (that is, they become "disconnected" everlastingly). Furthermore we have described countermeasures that allow thwarting these attacks.

The authors of SAKE$^+$ and SAKE$^+$-AM build their work upon the protocols proposed in [5]. They make several changes and additions in order to improve the latter. Our results show that these "enhancements" do not achieve their intended goals, and actually decrease the security of the resulting protocols.

Finally our results question the correctness of the security proofs provided in [1], made in the computational model (using the game-based methodology [8, 27]), and with the ProVerif verification tool [12].

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, 20200705:211635 (2020), https://eprint.iacr.org/2020/778
2. Alt, S., Fouque, P.A., Macario-Rat, G., Onete, C., Richard, B.: A cryptographic analysis of UMTS/LTE AKA. In: Manulis, M., Sadeghi, A.R., Schneider, S. (eds.) ACNS 16. LNCS, vol. 9696, pp. 18–35. Springer, Heidelberg (Jun 2016). https://doi.org/10.1007/978-3-319-39555-5_2
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). https://doi.org/10.2478/popets-2019-0065
4. Avoine, G.: Adversarial model for radio frequency identification. Cryptology ePrint Archive, Report 2005/049 (2005), http://eprint.iacr.org/2005/049
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). https://doi.org/10.1007/978-3-030-40186-3_10
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). https://doi.org/10.1007/978-3-642-16822-2_13
7. 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). https://doi.org/10.1007/3-540-48329-2_21
8. 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). https://doi.org/10.1007/11761679_25
9. Bhargavan, K., Blanchet, B., Kobeissi, N.: Verified models and reference implementations for the TLS 1.3 standard candidate. In: 2017 IEEE Symposium on Security and Privacy. pp. 483–502. IEEE Computer Society Press (May 2017). https://doi.org/10.1109/SP.2017.26
10. Bhargavan, K., Boureanu, I., Delignat-Lavaud, A., Fouque, P.A., Onete, C.: A formal treatment of accountable proxying over TLS. In: 2018 IEEE Symposium on Security and Privacy. pp. 799–816. IEEE Computer Society Press (May 2018). https://doi.org/10.1109/SP.2018.00021
11. Bhargavan, K., Boureanu, I., Fouque, P.A., Onete, C., Richard, B.: Content delivery over TLS: a cryptographic analysis of keyless SSL. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 1–16. IEEE (April 2017). https://doi.org/10.1109/EuroSP.2017.52
12. Blanchet, B., Smyth, B., Cheval, V., Sylvestre, M.: ProVerif 2.01: Automatic cryptographic protocol verifier, user manual and tutorial (April 2020)
13. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13**(7), 422–426 (July 1970). https://doi.org/10.1145/362686.362692
14. 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). https://doi.org/10.1007/978-3-662-49890-3_26

15. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (May 2001). https://doi.org/10.1007/3-540-44987-6_28
16. Canetti, R., Krawczyk, H.: Security analysis of IKE's signature-based key-exchange protocol. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 143–161. Springer, Heidelberg (Aug 2002). https://doi.org/10.1007/3-540-45708-9_10, http://eprint.iacr.org/2002/120/
17. Cremers, C., Horvat, M., Hoyland, J., Scott, S., van der Merwe, T.: A comprehensive symbolic analysis of TLS 1.3. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1773–1788. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134063
18. Dierks, T., Rescorla, E.: The transport layer security (TLS) protocol – version 1.2 (August 2008), https://tools.ietf.org/html/rfc5246
19. 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). https://doi.org/10.1145/2674005.2674994
20. 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). https://doi.org/10.1109/EuroSP.2017.18
21. Fouque, P.A., Onete, C., Richard, B.: Achieving better privacy for the 3GPP AKA protocol. PoPETs **2016**(4), 255–275 (Oct 2016). https://doi.org/10.1515/popets-2016-0039
22. 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
23. 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 07. pp. 242–252. ACM Press (Mar 2007)
24. Ma, C., Li, Y., Deng, R.H., Li, T.: RFID privacy: relation between two notions, minimal condition, and efficient construction. In: Al-Shaer, E., Jha, S., Keromytis, A.D. (eds.) ACM CCS 2009. pp. 54–65. ACM Press (Nov 2009). https://doi.org/10.1145/1653662.1653670
25. Ouafi, K., Phan, R.C.: Privacy of recent RFID authentication protocols. In: Chen, L., Mu, Y., Susilo, W. (eds.) Information Security Practice and Experience, 4th International Conference, ISPEC 2008. LNCS, vol. 4991, pp. 263–277. Springer (2008). https://doi.org/10.1007/978-3-540-79104-1_19
26. Rescorla, E.: The transport layer security (TLS) protocol version 1.3 (August 2018), https://tools.ietf.org/html/rfc8446
27. 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
28. Vaudenay, S.: On privacy models for RFID. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 68–87. Springer, Heidelberg (Dec 2007). https://doi.org/10.1007/978-3-540-76900-2_5

A Detailed Scenario of the Disconnection Attack

In this section, we provide the detailed scenario of the disconnection attack presented in Section 4.5.

The adversary relies almost only on the peculiarities of SAKE$^+$ in order to achieve its goal.

In SAKE$^+$, it is possible for the initiator A to be in sync, or one step ahead, or one step behind with respect to the master keys shared with the responder B. SAKE$^+$ uses the same synchronisation technique as the SAKE protocol with respect to the master keys [5]. In SAKE, whatever the gap δ_{AB} between A and B when a new protocol run starts, the two parties are always able to resynchronise their master keys in the continuity of the session, and eventually to successfully complete the session (because $\delta_{AB} \in \{-1, 0, 1\}$). Yet, in SAKE$^+$, when the initiator A is one step ahead at the beginning of a new session, this desynchronises the two parties with respect to the *identity parameter* they share.

Let us assume that the current epoch with respect to A is t, and B still belongs to epoch $t-1$. That is, B stores

$$id_B = id_{B,t-1}, K' = K'_{t-1}, K = K_{t-1}$$

and A stores

$$\begin{gathered}(id_{B,j}, K'_j, K_j),\ (id_{B,j-1}, K'_{j-1}, K_{j-1}),\ (id_{B,j+1}, K'_{j+1}, K_{j+1})\\ =\\ (id_{B,t}, K'_t, K_t),\ (id_{B,t-1}, K'_{t-1}, K_{t-1}),\ (id_{B,t+1}, K'_{t+1}, K_{t+1})\end{gathered}$$

First session. Upon reception of a fresh initial message sent by A, B computes the message m_B with $K' = K'_{t-1}$ (and $id_B = id_{B,t-1}$ or some pseudo-random value r_α depending on the value of ϕ). When A receives m_B it can correctly verify the MAC tag τ_B with K'_{t-1}. Therefore, A does not update the parameters (authentication and derivation master keys, and identity) corresponding to B. It responds with the message m_A (which includes $\epsilon = 1$). When it receives m_A, B updates its parameters twice with the function defined in SAKE$^+$. That is, first

$$\begin{aligned} id_B &\leftarrow \mathsf{update}(id_B \| K') = id_{B,t}\\ K' &\leftarrow \mathsf{update}(K') \qquad = K'_t\\ K &\leftarrow \mathsf{update}(K') \qquad = K_t \end{aligned}$$

and then

$$\begin{aligned} id_B &\leftarrow \mathsf{update}(id_B \| K') = id_{B,t+1}\\ K' &\leftarrow \mathsf{update}(K') \qquad = K'_{t+1}\\ K &\leftarrow \mathsf{update}(K') \qquad = K_{t+1} \end{aligned}$$

and B sets ϕ to 0. Then B computes the fourth message of the protocol run $(id_B \| \tau'_B)$ with its current parameters with $id_B = id_{B,t+1}$, and $\tau'_B = \mathsf{Mac}(K', r_B \| r_A) = \mathsf{Mac}(K'_{t+1}, r_B \| r_A)$.

When A receives this message, it can correctly verify its MAC tag with $K'_{j+1} = K'_{t+1}$. Therefore, A does the same update operations *as in the SAKE* *protocol*. That is, A updates the master keys corresponding to B, but *not the identity parameters*. The entry stored by A and related to B is changed from

$$(id_{B,t}, K'_t, K_t), (id_{B,t-1}, K'_{t-1}, K_{t-1}), (id_{B,t+1}, K'_{t+1}, K_{t+1})$$

into

$$(id_{B,t}, K'_{t+1}, K_{t+1}), (id_{B,t-1}, K'_t, K_t), (id_{B,t+1}, K'_{t+2}, K_{t+2}).$$

Party A computes the last message of the protocol with the now current parameters $(t+1)$: $id_{B,t}$ and K'_{t+1}. To that point, there are two options:

- if B verifies the identity carried in m'_A, the message is rejected (because $id_{B,t} \neq id_B = id_{B,t+1}$), and the session aborts;
- if B verifies only the MAC tag, the session ends successfully (because $K' = K'_{t+1}$).

Nevertheless, this does not change the parameters on A's and B's side.

Second session. When A starts a new session with B, the latter responds with a fresh message m_B computed with $K' = K'_{t+1}$, and $\phi = 0$. Hence m_B includes the current value $id_B = id_{B,t+1}$. With this message, A finds a match in its database. In this matching entry, $id_{B,t+1}$ refers to K'_{t+2}, but the entry includes also to K'_{t+1} (associated to $id_{B,t}$).

Remark 8. To that point, we recall our observation made in Remark 1, Section 2.1. That is, since id_B and K' evolve the same way (on B's side), id_B could be used as key identifier in addition to as (ephemeral) party identifier. Nonetheless, the description given in [1], Section 6.2, explicitly indicates that the different master authentication keys corresponding to the matching entry in A's database are tested by A. Hence, we stick to this processing.

That being said, if A uses K'_{t+2} to verify the MAC tag τ_B, the result is `false`, and A aborts. This does not change what happens next in this scenario. This delays only the conclusion by one additional session. That is, what is described below happens in a third session instead of this second session.

Indeed, if A aborts prematurely, B does not receive the message m_A, hence ϕ remains set to 1. Eventually, A starts a new session, and B responds with a message of the form $m_B = r_\alpha \| r_B \| \tau_B$ computed with $K' = K'_{t+1}$ (because B has not updated its master keys). Upon reception of this message, A does not find a match to r_α in its database. Hence it tries all existing authentication keys, and finds K'_{t+1}. Then, the same operations as those described below are executed.

The MAC tag is correctly verified with K'_{t+1} (i.e., A is in sync with B with respect to the master keys). Then A computes the message m_A with $\epsilon = 0$ and K'_{t+1} (because K'_{t+1} is the authentication master key of the current epoch for A). Finally, A updates all the parameters corresponding to B:

$$(id_{B,t}, K'_{t+1}, K_{t+1}), (id_{B,t-1}, K'_t, K_t), (id_{B,t+1}, K'_{t+2}, K_{t+2})$$

is changed into

$$(\tilde{id}_{B,t+1}, K'_{t+2}, K_{t+2}), (\tilde{id}_{B,t}, K'_{t+1}, K_{t+1}), (\tilde{id}_{B,t+2}, K'_{t+3}, K_{t+3})$$

with $\tilde{id}_{B,i} = \mathsf{update}(id_{B,i-1} \| K'_i)$, $i \in \{t, t+1, t+2\}$.

When B receives m_A, it can correctly verify the message with its authentication key $K' = K'_{t+1}$. Hence, it updates its own parameters:

$$\begin{aligned} id_B &\leftarrow \mathsf{update}(id_B \| K') &&= \mathsf{update}(id_{B,t+1} \| K'_{t+1}) &&= id_{B,t+2} \\ K' &\leftarrow \mathsf{update}(K') &&= \mathsf{update}(K'_{t+1}) &&= K'_{t+2} \\ K &\leftarrow \mathsf{update}(K) &&= \mathsf{update}(K_{t+1}) &&= K_{t+2} \end{aligned}$$

and sets ϕ to 0. Party B computes message m'_B with $id_B = id_{B,t+2}$ and $K' = K'_{t+2}$.

Upon reception of m'_B by A, there are two options:

– if A verifies the identity parameter carried in m'_B, it rejects the message because, with high probability, it finds no match to $id_B = id_{B,t+2}$ in its database (in particular $id_B \neq \tilde{id}_{B,t+1}$), and the session aborts;
– if A verifies only the MAC tag, the message is deemed as valid, and A responds with m'_A computed with $\tilde{id}_{B,t+1}$ and K'_{t+2}. Then, depending on B's behaviour with respect to m'_A, either the session aborts or it ends correctly (see above, first session).

In any case, the parameters remain unchanged respectively on A's and B's side.

Third and next sessions. To that point, with high probability, $id_B \neq \tilde{id}_{B,i}$, $i \in \{t, t+1, t+2\}$. That is, A and B are desynchronised with respect to the identity parameter.

When A starts a new session, the identity parameter sent by B in $m_B = id_B \| r_B \| \tau_B$ (with $\tau_B = \mathsf{Mac}(K', id_B \| id_A \| r_B \| r_A)$) has no match in A's database. Therefore, A tries the different authentication master keys, and the corresponding identity parameter found in its database. Since $id_B \neq \tilde{id}_{B,i}$, $i \in \{t, t{+}1, t{+}2\}$, the verification of the MAC tag τ_B yields always `false`. The same recurs in the next sessions. Hence A and B are unable to communicate anymore.

In SAKE$^+$-AM, the responder (resp. initiator) updates the master keys and the identity parameters in the same way as the initiator (resp. responder) in SAKE$^+$. The same sequence of operations as in SAKE$^+$ is executed. Therefore, the same scenario can be applied against SAKE$^+$-AM.