A Cryptographic Analysis of the 3GPP AKA Protocol

Stephanie Alt[1], Pierre-Alain Fouque[2], Gilles Macario-rat[4], Cristina Onete[3] and Benjamin Richard[4]

[1] DGA Bruz, France , s.alt@free.com
[2] IRISA, University of Rennes 1, France, pierre-alain.fouque@ens.fr
[3] INSA / IRISA Rennes, France, cristina.onete@gmail.com
[4] Orange Labs, Chatillon, France ,
gilles.macariorat@orange.com,benjamin.richard@orange.com

Abstract. Secure communications between mobile subscribers and their associated operator networks require mutual authentication and key derivation protocols. The 3GPP standard provides the AKA protocol for just this purpose. Its structure is generic, to be instantiated with a set of seven cryptographic algorithms. The currently-used proposal instantiates these by means of a set of AES-based algorithms called MILENAGE; as an alternative, the ETSI SAGE committee submitted the TUAK algorithms, which rely on a truncation of the internal permutation of Keccak.
In this paper, we provide a formal security analysis of the AKA protocol in its complete three-party setting. We formulate requirements with respect to both Man-in-the-Middle (MiM) adversaries, i.e. key-indistinguishability and impersonation security, and to local untrusted serving networks, denoted "servers", namely state-confidentiality and soundness. We prove that the unmodified AKA protocol attains these properties as long as servers cannot be corrupted. Furthermore, adding a unique server identifier suffices to guarantee all the security statements even in in the presence of corrupted servers. We use a modular proof approach: the first step is to prove the security of (modified and unmodified) AKA with generic cryptographic algorithms that can be represented as a unitary pseudorandom function –PRF– keyed either with the client's secret key or with the operator key. A second step proceeds to show that TUAK and MILENAGE guarantee this type of pseudorandomness, though the guarantee for MILENAGE requires a stronger assumption. Our paper provides (to our knowledge) the first complete, rigorous analysis of the original AKA protocol and these two instantiations. We stress that such an analysis is important for any protocol deployed in real-life scenarios.

Keywords: security proof, AKA protocol, TUAK, MILENAGE.

1 Introduction

A secure, symmetric, authenticated key-exchange (or key-agreement) protocol is usually built in two phases. During the first phase, the parties authenticate each other and exchange some master key. This master key is then used to derive one or multiple secret keys, as well as other useful values. In a second phase,

these derived keys are used to construct a secure channel between the parties, allowing them to exchange data while providing confidentiality, integrity, and data authentication.

In this paper, we focus on the Authentication and Key Agreement protocol (AKA) used in 3G and 4G networks, more specifically the 3G UMTS AKA (Universal Mobile Telecommunications System) and 4G EPS AKA (Evolved Packet System) protocol. The AKA protocol is used in a greater context in the 3rd Generation Partnership Project (3GPP), which aims to develop the specifications for next generation mobile systems. The Technical Specifications 33 (TS 33) and 35 (TS 35) cover the security and privacy aspects of the new system, from both an architectural and a security algorithm standpoint. The greater context in which this protocol is meant to be used makes a thorough security analysis imperative. We note that the protocol does not exactly follow classical symmetric key-agreement designs; for instance, one of its peculiarities is that, while clients (also called subscribers), are each associated with an individual secret key, all clients serviced by the same operators also share this operator's key (which is nevertheless never stored in clear on the client machines).

In this paper we focus on the provable security of this protocol. Our analysis closely follows its specifications, notably the set of seven functions which generate the cryptographic output necessary to attain security in the AKA protocol. In the original seven-algorithm proposal called MILENAGE [1], these functions relied on AES encryption. As an alternative to MILENAGE, another set of algorithms called TUAK [2] was proposed, the latter relying on a truncation of Keccak's internal permutation. The winner of the SHA-3 hash function competition, Keccak relies on the sponge construction [10], thus offering both higher performance, in hardware and software, than AES, and resistance to many generic attacks. While the TUAK algorithms, designed by the ETSI SAGE group, inherit Keccak's superior performance, they do not use the Keccak permutation in a usual, black box way. Instead the internal permutation is truncated, then used in a cascade, which makes it non-trivial to analyze. We cannot simply use the same assumptions for the truncated version as we would for the original permutation, either. Our analysis of the key security, as well as client- and respecticaly server-impersonation resistance of the protocol concerns both the classical MILENAGE-based version, and the one using TUAK.

Related Work. Bellare and Rogaway first proposed a security model for authentication and key exchange mechanisms in [15], also in a symmetric setting. By assuming only the existence of pseudo-random functions, they propose constructions and prove their security in the model. Their framework was later extended with the contribution of Pointcheval [14]. In addition [16] proposes a definition of security of key-exchange protocols relying on an "ideal third party" approach. A further model for generic session-oriented protocols was proposed in [17] and extended in [23]. Although we use Bellare, Pointcheval, and Rogaway methodologies in our analysis, we cannot simply "import" their model, as we explain in more detail below.

Few papers give a security proof for the AKA protocol, especially when instantiated with MILENAGE. The closest results to a security proof – see below – use automated (formal) verification. While this approach has many advantages, and an automated proof is a good first step towards a thorough security analysis, one important disadvantage is that automated verification does not give an exact reduction to the security of the underlying primitives; thus, the proof statement is not easy to quantify, making it hard to approximate the tightness of the proof and the size of the parameters. In this sense, our results are stronger.

As far as we know, only two papers clearly focus on the mutual authentication and key-secrecy properties of AKA. A first one [18] points out some problems with using sequence numbers and outlines a corrupted-network redirection attack, but does not attempt a security proof. Note that the AKA protocol is *à priori* designed with the assumption that the operator trusts the network; the network attack described by [18] falls outside the scope of our paper as we do not consider how networks and operators implement the protocol. Furthermore, note that adding a simple network-specific constant in the computation of the MAC algorithms should prevent such attacks. By removing the sequence number, the same authors propose a stateless variant called AP-AKA, with a security proof based on Shoup's formal model [25].

A second paper [7] refers to the authentication and key security of AKA, but focuses mainly on client privacy. They attempt to do an automated verification proof for the AKA protocol, using ProVerif [8]; however, they are only able to assess a *modified* version, which randomizes the sequence number. Since this modification is fundamental, their results cannot be applied to the original protocol. In order to better model the true sequence number updates in the protocol, we have used an extension of ProVerif called StatVerif [20], which was proposed by Arapinis et al. to handle automatic verification for protocols with global state. We were, however, unable to use this tool towards giving a formal security proof for AKA, as we discuss in Appendix F.1.

While Gilbert provides an out-of-context security proof for the MILENAGE algorithms [13], showing they operate as a kind of counter mode in deriving key materials (MILENAGE runs AES multiple times as a one block to many blocks expansion function), it is unclear whether the proved indistinguishablilty properties are strictly useful to guarantee the security of the full AKA protocol. By contrast, we begin by analyzing the security of the AKA scheme, and are able to show that the security statements hold when it is instantiated both with MILENAGE and with TUAK.

Our contributions. In this paper we give the first full and rigorous cryptographic security analysis of the AKA protocol, specifically with respect to key-secrecy and mutual authentication (for a MiM adversary) and key-confidentiality and soundness (for a malicious server adversary). This result is all the more significant since even fifteen years after its proposal, the security of the AKA protocol is not well understood, in spite of its importance in the future of mobile communications. Not only has no formal cryptographic proof ever been given for the AKA scheme, but even the automated verification results in the literature

only concern modifications of the original scheme. This is mostly due to the fact that the AKA construction is stateful and that its mutual authentication guarantee is asymmetric in the sense that it offers more client-impersonation than server-impersonation resistance. We note that a formal cryptographic analysis of AKA is also essential in order to understand how far the properties of the underlying primitives (MILENAGE or TUAK) guarantee the security of the full protocol.

Though AKA may seem to be a typical – if stateful – symmetric key agreement protocol, its design is convoluted and includes several unusual features. The sequence numbers provide state to the protocol, and are tied to a resynchronization procedure. The server authentication step allows an unorthodox kind of relay attack, which permits a degree of server impersonation. Furthermore, clients registered with the same operator share that operator's key $\mathsf{sk}_{\mathsf{Op}}$, though not their individual client keys. An interesting fact regarding the operator key is that it is never stored in clear in the client's SIM card. Finally, clients and servers may become desynchronized, and the resynchronization procedure does introduce a further protocol step. Due to these features, we cannot use a classical Bellare-Rogaway [15] or Bellare-Pointcheval-Rogaway [14] model for our analysis, though we employ a modified version of it. Our model is robust with respect to multiple clients, multiple operators, and different types of corruptions, and we consider re-synchronizations.

We prove two different, strong results. We first prove the security of the AKA protocol assuming the pseudorandomness of a generic PRF called G, which outputs the session keys and the authentication material. Thus, this first result is more generic than analyzing the individual TUAK- and MILENAGE-based versions of the protocol; in fact, what we show is that *any* instantiation of the protocol which can be modeled in this way will retain the same security properties.

Our second strong result is to show that the TUAK and MILENAGE algorithms can be viewed as an instantiation of the generic pseudorandom function G (in the case of MILENAGE in fact we use two generic functions, but which are alternated). This is non-trivial, since the AKA protocol employs seven algorithms which use related input. We show how to generalize the algorithms to one, resp. two functions, and we rely on the fact that the output is independent in order to prove key secrecy and mutual authentication. The consequence is that the AKA construction attains these properties when instantiated both with TUAK and with MILENAGE.

Our results indicate, in a nutshell, that:

– The protocol (in which the (MILENAGE or TUAK) cryptographic functions are replaced by a function G) offers key-secrecy – the property that session keys are indistinguishable from random – if G is a pseudo-random function. We note that a PRF assumption is also used by [22] to prove the security of the TLS key derivation. There is a fine subtlety in the quantification of the attacks, which have to take into account the number of instances of the two parties, but also the number of resynchronizations per instance. While

in our security model we take into account the fact that operator keys are not leaked by corruptions (since they are not stored in clear on the SIM card and are hard to retrieve), we also prove the same security guarantee even if the operator keys leak[1].
- We prove that the protocol guarantees a stronger degree of impersonation resistance for clients than for servers. This can be done by modelling the notion of time into our security game and distinguishing between online and offline relays. We show a clear separation between the client- and server-impersonation models by describing a concrete server impersonation attack against the AKA protocol, relying on offline relays. Thus, while AKA is client-impersonation-resistant with respect to adversaries which may do offline relays, it does not attain the same degree of server-impersonation resistance. We model both properties and show that the security of the protocol can be reduced to the PRF assumption for the function G.
- Alternatively, we also prove client- and impersonation-security in a more classical model, which does not feature time. For this model, however, we are not able to fully capture the stronger client-impersonation guarantee, and thus we only prove a weaker one, equivalent to the server-impersonation statement.
- We prove that the MILENAGE algorithms used by AKA offer indeed the property we require, namely pseudorandomness under the well-known assumption that the Advanced Standard Encryption is a good pseudo-random function [9]. Note that a such assumption is also used to prove the different modes of operations based on AES.
- We also prove that the TUAK algorithms offer the same pseudorandomness property, under the well-known assumption that the internal $\mathsf{f}_{\mathsf{Keccak}}$ permutation is a good permutation, which implies that a truncated keyed version of this permutation is a good pseudo-random function [9]. This assumption is reasonable since the indifferentiability proof of the Sponge construction relies on the same assumption [10]. Note that the TUAK algorithms employed by the AKA protocol are constructed as a non-orthodox cascade of two iterations of the Keccak permutation, which is done in an unusual way, unlike the construction used in the Duplex mode of operation in order to construct authenticated cipher [11].

While the result of this paper is mostly a positive one (the protocol can be proved secure), there are a few problematic points that we discuss in our Lessons Learned section. A specific feature of AKA is the use of a sequence number, which is updated at each successful authentication of the other party, and which can be resynchronized if necessary. This sequence number enters as input in computing the server authentication string, and acts the part of the client's randomness (since it changes regularly). While it permits server authentication, the sequence number represents a long-term state variable, and the updating

[1] In fact, paradoxically, leaking the operator keys makes our proofs easier; otherwise, it is tricky to simulate consistent operator keys for clients other than the target client, but who share the same operator as the latter.

mechanism does not provide as much entropy as choosing a random number. The sequence number also permits sessions to be related to each other in a specific way, which is captured in our model by the notion of freshness. Another problematic point is sharing the operator key between all the clients of a same operator. Though the key is not saved in clear, we need to consider the corruption of the operator keys.

Finally we note that, in practice, the user answers to the initial identification request by sending (in clear) either its International Mobile Subscriber Identity IMSI or a Permanent Mobile Subscriber Identity TMSI. As specified in some studies as [21], using these values cannot guarantee confidentiality, nor user privacy. In particular this yields a server-impersonation vulnerability, which we use as a separating counterexample between the models for the client- and server-impersonation resistance. We note that one easy way to mitigate our separating counterexample would be to generate a pseudorandom UID value, possibly by using public key encryption mechanisms (only the server would need to certify the key in this scenario).

We discuss several such protocol issues in Section 6.

2 The AKA protocol

2.1 Notations

Notation. Throughout the rest of the document, we will consider for a bitstring x, $|x|$ for the bit-size of x and $\lfloor x \rfloor_{i..j}$ for the bitstring from position i to j of the initial bitstring x. If f is a function, then $y \leftarrow f(x)$ means that the y is the output of f when run on input x. Therefore, $y \xleftarrow{\$} \{0,1\}^n$ means that the value y is chosen uniformly in the set $\{0,1\}^n$. For bit strings x and y, we denote $x \| y$ the concatenation between x and y. We denote $\oplus$ the bitwise exclusive-or operation. For one bit b, we write b^n to denote a n-bit string composed of a concatenation of n bits b. Therefore, we denote λ the empty message. Finally, we denote $*$ the value suggesting that the entity sends no messages and λ the value suggesting that the entity receives no messages.

2.2 Description of the AKA protocol

In mobile telecommunications, 3G networks use a variant of AKA which is fully depicted in Figure 6. Once the protocol is run, the client and server output session keys (CK, IK), which are then used to secure future message-exchanges. The same protocol serves as the backbone of the 4G LTE protocol, with the following differences. First, the client is identified by means of a different identifier called a GUTI (more details in [6]), as opposed to the tuple of permanent and temporary identifiers we describe below. For the purposes of our analysis, however, the nomenclature that is used in such networks is not relevant. A more significant difference is the fact that instead of using CK and IK as session keys, the client

and server use a key that is *derived* from these two keys, by means of a key derivation function KDF[2] and only this latter is shared with the server. In terms of security, this would make our security statements also depend on the security of the key-derivation function KDF.

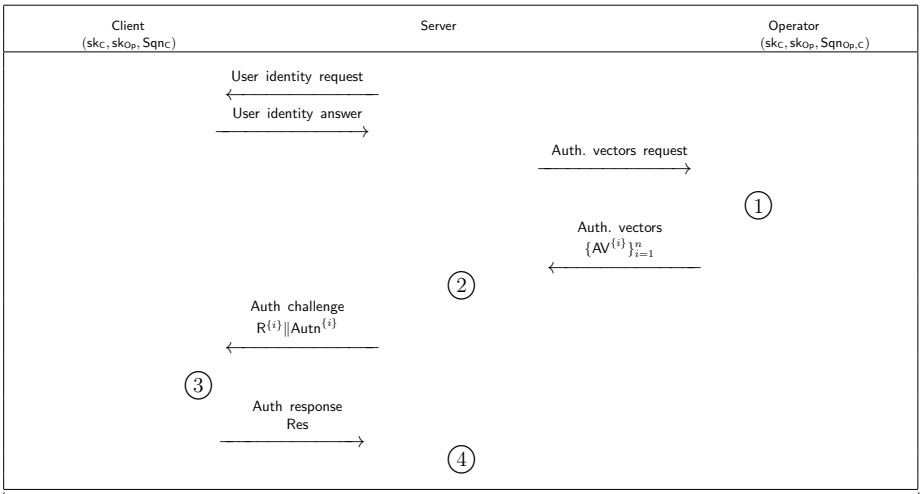
This protocol features two main *active* actors: the client (in 3GPP terminology $\mathsf{ME}/\mathsf{USIM}$) and the server (denoted VLR). The third, only selectively-active party is the operator (denoted HLR). The tripartite setup of AKA was meant for roaming, in which case the server providing the mobile coverage is not the client's operator, and may be subject to different legislation and vulnerabilities than the latter. Thus, although the server is trusted to provide services across a secure channel, it must not learn long-term sensitive information about either clients or their home operators. Using the server as a mere proxy would be an ideal solution; however, the server/operator communication is (financially) expensive.

Section 3 describes in detail the setup of the three parties. Clients C and operators Op require both the client's secret key sk_C and the operator's secret key $\mathsf{sk}_{\mathsf{Op}}$[3]. The client and operator also keep track of sequence numbers Sqn_C (resp. $\mathsf{Sqn}_{\mathsf{Op},\mathsf{C}}$), ideally kept close to each another and updated after each successful authentication. The updating procedure is quite simple and predictable, e.g. incrementing the counter by a fixed value. If the two values are too far apart, the client initializes a re-synchronization procedure. The three parties: clients, servers, and operators, also know the client's permanent identifier IMSI. Clients and servers moreover keep track of tuples (IMSI, TMSI, LAI), the last two values forming a unique temporary identifier, which is updated at every session.

The AKA protocol, depicted in Figure 6, proceeds in several subparts. The first two protocol exchanges are between the client C and the server S over an *insecure channel* and they make up the *user identification* step. At the end of this phase, the server will associate C with an identifier, either the permanent International Mobile Subscriber Identity IMSI or a tuple consisting of a Temporary Mobile Subscriber Identity TMSI and the Local Area Identifier LAI of the server that issued the latest TMSI. The exact way the identification proceeds is vital to the client's privacy; however, in this paper we focus only on the *security* of AKA and associate each client with a unique user ID UID per client. We explain this approach in more detail at the end of this section. Once the server can associate the client with an identifier UID, it proceeds either to the *authentication vector generation* step (detailed in the set ① of instructions in Fig. 6), or to the *authenticated key-exchange* part (detailed in instruction sets ②-④). The former of these is run by the server and the operator of the client C over a *secure channel*, and it provides the server S with authentication and key-exchange material for a batch of AKA sessions with C; whenever S runs out of AKE material, it re-runs the vector generation step. For each

[2] To be more precise, this key is usually denoted as $\mathsf{K}_{\mathsf{asme}}$, and is computed as follows: $\mathsf{K}_{\mathsf{asme}} = \mathsf{KDF}(\mathsf{CK}\|\mathsf{IK}, \mathsf{ID}_{\mathsf{SN}}, \mathsf{Sqn} \oplus \mathsf{AK}, \mathsf{const})$, with $\mathsf{ID}_{\mathsf{SN}}$ the identity of the serving operator network.

[3] Technically speaking, the client never stores this value in clear; instead it uses a pseudorandom value Top_C computed from the client and operator keys.



Instructions:

Client	Server	Operator
3: Compute AK using $\mathsf{R}^{\{i\}}$. Recover $\mathsf{Sqn}^{\{i\}}$ (from AK). Check $\mathsf{Mac_S}$ value. If $\mathsf{Sqn}^{\{i\}} \in (\mathsf{Sqn_C}, \mathsf{Sqn_C} + \Delta)$: Compute: $\mathsf{CK} \leftarrow \mathcal{F}_3(\mathsf{sk_C}, \mathsf{sk_{Op}}, \mathsf{R}^{\{i\}})$, $\mathsf{IK} \leftarrow \mathcal{F}_4(\mathsf{sk_C}, \mathsf{sk_{Op}}, \mathsf{R}^{\{i\}})$, Set $\mathsf{Res} := \mathcal{F}_2(\mathsf{sk_C}, \mathsf{sk_{Op}}, \mathsf{R}^{\{i\}})$. Update $\mathsf{Sqn_C} := \mathsf{Sqn}^{\{i\}}$. Else *re-synchronization*	2: Store $\{\mathsf{AV}^{\{i\}}\}_{i=1}^{n}$. Choose $\mathsf{AV}^{\{i\}}$ one by one in order. Then it sends the related challenge. 4: $\mathsf{Res} \stackrel{?}{=} \mathsf{Mac_C}$.	1: For each $i = 1, \ldots, n$, compute: Generate $\mathsf{R}^{\{i\}}$. Compute: $\mathsf{Sqn}^{\{i\}} \leftarrow \mathsf{inc}(\mathsf{Sqn_{Op,C}})$ $\mathsf{Mac_S^{\{i\}}} \leftarrow \mathcal{F}_1(\mathsf{sk_C}, \mathsf{sk_{Op}}, \mathsf{R}^{\{i\}}, \mathsf{Sqn}^{\{i\}}, \mathsf{AMF})$, $\mathsf{Mac_C^{\{i\}}} \leftarrow \mathcal{F}_2(\mathsf{sk_C}, \mathsf{sk_{Op}}, \mathsf{R}^{\{i\}})$, $\mathsf{CK}^{\{i\}} \leftarrow \mathcal{F}_3(\mathsf{sk_C}, \mathsf{sk_{Op}}, \mathsf{R}^{\{i\}})$, $\mathsf{IK}^{\{i\}} \leftarrow \mathcal{F}_4(\mathsf{sk_C}, \mathsf{sk_{Op}}, \mathsf{R}^{\{i\}})$, $\mathsf{AK}^{\{i\}} \leftarrow \mathcal{F}_5(\mathsf{sk_C}, \mathsf{sk_{Op}}, \mathsf{R}^{\{i\}})$, $\mathsf{Autn}^{\{i\}} \leftarrow (\mathsf{Sqn}^{\{i\}} \oplus \mathsf{AK}^{\{i\}}), \mathsf{AMF}, \mathsf{Mac_S^{\{i\}}}$. $\mathsf{AV}^{\{i\}} := \mathsf{R}^{\{i\}}, \mathsf{CK}^{\{i\}}, \mathsf{IK}^{\{i\}}, \mathsf{Autn}^{\{i\}}, \mathsf{Mac_C^{\{i\}}}$, with $\mathsf{Sqn_{Op,C}} = \mathsf{Sqn}^{\{i\}}$. End For.

Fig. 1. The AKA Procedure.

session, Op prepares an authentication vector AV consisting of: a fresh random value R; a server-authentication string $\mathsf{Mac_S}$ (authenticating R and the value $\mathsf{Sqn_{Op,C}}$); a client-authentication string $\mathsf{Mac_C}$ (authenticating R only); the session keys CK and IK; and a one-time-pad encryption of $\mathsf{Sqn_{Op,C}}$ with a pseudorandom string AK. The values $\mathsf{Mac_S}, \mathsf{Mac_C}, \mathsf{CK}, \mathsf{IK}, \mathsf{AK}$ are output by cryptographic algorithms denoted $\mathcal{F}_1, \ldots, \mathcal{F}_5$ respectively. The AKA protocol also provides the client with the algorithms $\mathcal{F}_1^*, \mathcal{F}_5^*$ for re-synchronization. All algorithms take as input the client's secret key $\mathsf{sk_C}$, the operator's key $\mathsf{sk_{Op}}$ and the random value R; in addition, $\mathcal{F}_1$ and $\mathcal{F}_1^*$ also use the operator's and resp. the client's sequence number. The server is given a batch of vectors of the form: $\mathsf{AV} = (\mathsf{R}, \mathsf{CK}, \mathsf{IK}, \mathsf{Mac_S}, \mathsf{Mac_C}, \mathsf{AMF}, \mathsf{AK} \oplus \mathsf{Sqn_{Op,C}})$, in which AMF is a public authentication management field managed by the operator.

The *authenticated-key-exchange* step allows clients and servers to mutually authenticate and compute session keys over an *insecure channel*. The server chooses the next AV from the latest batch, using the random R and the string $\mathsf{Autn} = (\mathsf{Sqn_{Op,C}} \oplus \mathsf{AK}) \| \mathsf{AMF} \| \mathsf{Mac_S}$ as a challenge. The client uses R to compute AK and recover $\mathsf{Sqn_{Op,C}}$. If the received $\mathsf{Mac_S}$ verifies and $\mathsf{Sqn_{Op,C}}$ is within a predefined distance Δ of $\mathsf{Sqn_C}$, then C computes $(\mathsf{CK}, \mathsf{IK})$ and the value $\mathsf{Mac_C}$,

sending this last value to S. If the two sequence numbers are too far apart, then C forces a re-synchronization, which we describe below. If no re-synchronization is needed, the client updates $\mathsf{Sqn_C}$ to $\mathsf{Sqn_{Op,C}}$, and S verifies the received authentication value with respect to the $\mathsf{Mac_C}$ sent by Op. If $\mathsf{Mac_C}$ verifies, then S sends an acknowledgement to Op and runs the TMSI *re-allocation*.

If $\mathsf{Sqn_{Op,C}}$ is too far from $\mathsf{Sqn_C}$, an optional *re-synchronization* is run by all three parties. The client uses $\mathsf{Sqn_C}$ to compute values $\mathsf{Mac_S^*}$ and $\mathsf{AK^*} \oplus \mathsf{Sqn_C}$ as the operator did, using the session R, but algorithms $\mathcal{F}_1^*$ and $\mathcal{F}_5^*$ rather than $\mathcal{F}_1$ and $\mathcal{F}_5$. If $\mathsf{Mac_S^*}$ verifies, Op resets $\mathsf{Sqn_{Op,C}}$ to $\mathsf{Sqn_C}$ and sends to S another batch of AV as before. The protocol restarts.

Finally, following any successful key-establishment, the server and the client go through the TMSI *re-allocation* step. The server sends an (unauthenticated) encryption of a new, randomly chosen TMSI value (which is unique per server) to the client C, using the key CK the parties agreed on. Encryption is done by means of the A5/3 algorithm [5], run in cipher mode. The new TMSI value, called $\mathsf{TMSI_{new}}$, is only permanently saved by S if acknowledged by the client; else, both values $\mathsf{TMSI_{new}}$ and the old $\mathsf{TMSI_{old}}$ are retained and can be used in the next authentication procedure.

Identities and reallocation. In our security analysis, we stick close to the original AKA protocol. However, one simplification we make throughout this paper is associating each prover with a single, unique UID, which we consider public. In practice, this identifier is the user's IMSI, which can be requested by servers in case a TMSI value is not traceable to an IMSI. From the point of view of security, any attack initiated by mismatching TMSI values (i.e. replacing one value by another) is equivalent to doing the same thing with IMSI values.

Another important part of the AKA protocol that we abstract in this analysis is the TMSI reallocation. If the TMSI system were flawless (a newly-allocated TMSI is reliable and non-modifiable by an active MiM), then we could prove a stronger degree of server impersonation than in our current model. As we discuss in Section 3, an active MiM can inject false TMSI values, which make servers request an IMSI value; if the MiM reuses this value, it can use a type of offline relay to impersonate the server. In particular, the use of the TMSI is undone by the back door allowing servers to demand the IMSI; simultaneously, insecurities in using TMSIs translate to the identification by IMSI.

3 Security model

3.1 Notations

Notation. Throughout the rest of the document, we will consider for a bitstring x, $|x|$ for the bit-size of x and $\lfloor x \rfloor_{i..j}$ for the bitstring from position i to j of the initial bitstring x. If f is a function, then $y \leftarrow f(x)$ means that the y is the output of f when run on input x. Therefore, $y \xleftarrow{\$} \{0,1\}^n$ means that the value y is chosen uniformly in the set $\{0,1\}^n$. For bit strings x and y, we denote $x \| y$ the

concatenation between x and y. We denote $\oplus$ the bitwise exclusive-or operation. For one bit b, we write b^n to denote a n-bit string composed of a concatenation of n bits b. Therefore, we denote λ the empty message. Finally, we denote $*$ the value suggesting that the entity sends no messages and λ the value suggesting that the entity receives no messages.

3.2 Key-indistinguishability and impersonation

The security goals of the AKA protocol are: the secrecy of the established sessions keys against both passive and active MiM adversaries, as well as mutual authentication. In particular, this protocol cannot guarantee (perfect) strong secrecy, as it uses symmetric long-term keys, which, once compromised, can also endanger past session keys. We formalize these goals in terms of three properties: key-indistinguishability, client-impersonation resistance, and server-impersonation resistance.

As mentioned in Section 1, these notions cannot be trivially proved in the Bellare-Rogaway model variations, e.g. [15,14]. Indeed, we need to propose a new model to take into account sequence numbers, resynchronizations, and a possible Man-in-the-Middle server-impersonation attack. Note that, even if this implies an imperfect mutual authentication, it has no impact on the secrecy (indistinguishability from random) of the sessions keys.

We split the guarantee of mutual authentication, which implies client and server impersonation resistance, into two properties. This is because the AKA protocol offers different degrees of security with respect to impersonation attacks for clients and for servers.

Setup and participants. We consider a set $\mathcal{P}$ of honest participants, which are either mobile clients C of the type ME/USIM subscribing to operators Op, or servers S. A participant is generically denoted as P. In all security games, the operators Op are black-box algorithms within the server S. We assume the existence of n_{C} clients, $\mathsf{n_S}$ servers and $\mathsf{n_{Op}}$ operators. If the operators are contained within the servers, we assume that all copies of the same operator are synchronized at all times.

Each client C is associated with a *unique* identifier UID, two long term static secret keys $\mathsf{sk_{UID}}$ (subscriber key), and $\mathsf{sk_{Op}}$ (operator key) which is common to all clients subscribing to a specific operator, and a long-term state $\mathsf{st_{UID}}$[4]. In particular, we consider multiple operators, with the restriction that each user may only be registered to a single operator[5]. In our model, we also assume for simplicity that the key space of all operators is identical, noting that neither the key-indistinguishability, nor the mutual authentication properties are affected

[4] The latter consists in practice of a sequence number $\mathsf{Sqn_{UID}}$, which is updated at each successful authenticated key exchange.

[5] We note that this seems to extend naturally to a case in which a single client may be registered with multiple operators, as long as the key-generation process for each operator is such that the registration of a single client to two operators is equivalent to representing a two-operator-client as two independent clients.

by the way operators choose their keys (the security of both the key exchange and the authentication properties rely just on the key *length*); the variation in the key space does, however, affect user privacy. Each operator is assumed to contain $\mathsf{sk}_{\mathsf{UID}}$ included in a database of tuples $(\mathsf{UID}, \mathsf{sk}_{\mathsf{UID}}, \mathsf{sk}_{\mathsf{Op}}, \mathsf{st}_{\mathsf{Op},\mathsf{UID}})$, each tuple corresponding to a single user of this operator. The last entry $\mathsf{st}_{\mathsf{Op},\mathsf{UID}}$ of each tuple denotes the long term state of the operator associated with that user – which may in fact differ from the state of the user itself. For the AKA protocol, the state is in fact a sequence number, associated with each client. Moreover, the servers do not contain any secret information of the operator or the subscriber.

In our model, each participant may run concurrent key-agreement executions of the protocol Π. We denote the j-th execution of the protocol by the party P as P_j. We tacitly associate each instance P_i with a session ID sid, a partner ID pid (consisting either of one or of multiple elements), and an accept/reject bit accept. As explained more in detail in Section 4, the partner ID is set to either the server or to a user identifier UID, whereas the session ID includes three values: the user ID given by the client (thus tacitly also the key associated with that UID), the randomness generated by the server, and the sequence number used for the authentication. Finally, the accept/reject bit is initialized to 0 and turns to 1 at the successful termination of the key-agreement protocol. We call this "terminating in an accepting state". In the absence of an adversary, the protocol is always run between a client C and a server S. For the AKA protocol, it is the server which begins the protocol by means of an ID request, and can thus be called its *initiator*, whereas the mobile client is the *respondent*. A successful termination of the protocol yields, for each party, a session key K (which for the AKA protocol consists of two keys), the session identifier sid, and the partner identifier pid of the party identified as the interlocutor. In AKA the client is authenticated by means of a challenge-response type of query, where the response is computed as a pseudo-random function of the key and (a part of) the challenge. The server is equally authenticated by means of an authentication string, also a pseudo-random function of the key, the challenge, and the long-term state that the server associates with that client. In particular, the challenge strings sent by the server are authenticated.

The notion of *key-indistinguishability* refers to the session keys calculated as a result of the key-exchange protocol (rather than to the long-term keys held by each party), requiring that they be indistinguishable from random bitstrings of equal length. The adversary MiM $\mathcal{A}$ can access instances of honest parties by means of oracles acting as interfaces; furthermore, $\mathcal{A}$ can schedule message deliveries, send tampered messages, or interact arbitrarily with any party, by means of the oracles below. We note that in the key-indistinguishability model the adversary may also know the long-term state (in our case, the sequence number) of both users and the server. This will also be the case in the impersonation games. Since the state is updated in a probabilistic way, we give the adversary a means of always learning the updated state of a party without necessarily corrupting it (the latter may rule out certain interactions due to notions of freshness, see below). Corruption is allowed and implied the related party is considered as *ad-*

versarially controlled. We use the same fundamental model, with similar oracles, also for the definitions of *client* and *server* impersonation.

We consider a finite (and public) list of n_{Op} operators $\mathsf{Op}_1, \ldots \mathsf{Op}_{n_{\mathsf{Op}}}$, for which the keys $\mathsf{sk}_{\mathsf{Op}_1}, \ldots \mathsf{sk}_{\mathsf{Op}_{n_{\mathsf{Op}}}}$ are generated independently and uniformly at random $\mathcal{S}_{\mathsf{op}}$.

Oracles. The adversary interacts with the system by means of the following oracles, in addition to a function G, which we model as a PRF.

- CreateCl(Op)$\rightarrow$ ($\mathsf{UID}, \mathsf{st}_{\mathsf{UID}}$): This oracle creates a client with unique identifier UID. Then the client's secret key $\mathsf{sk}_{\mathsf{UID}}$ and the sequence number $\mathsf{Sqn}_{\mathsf{UID}}$. The tuples ($\mathsf{UID}, \mathsf{sk}_{\mathsf{UID}}, \mathsf{sk}_{\mathsf{Op}}, \mathsf{Sqn}_{\mathsf{UID}}$) are associated with the client UID and with the corresponding operator Op (i.e. each "copy" of Op in each server does this). The operator sets $\mathsf{st}_{\mathsf{Op},\mathsf{UID}} := \mathsf{Sqn}_{\mathsf{UID}}$ and then keeps track of $\mathsf{st}_{\mathsf{Op},\mathsf{UID}}$. The adversary is given UID and $\mathsf{st}_{\mathsf{UID}}$.
- NewInstance(P) $\rightarrow$ (P_j, m): this oracle instantiates the new instance P_j, of party P, which is either a client or a server. Furthermore, the oracle also outputs a message m, which is either the first message in an honest protocol session (if P is a server) or $\perp$ (if P is a client). The state st of this party is initiated to be the current state of P, and it is initiated with the current value of TMSI, LAI.
- Execute($\mathsf{P}, i, \mathsf{P}', j$) $\rightarrow \tau$: creates (fresh) instances P_i of a server P and P'_j of a client, then runs the protocol between them. The adversary $\mathcal{A}$ receives the transcript of the protocol.
- Send(P, i, m) $\rightarrow m'$: simulates sending message m to instance P_i of P. The output is a response message m' (which is set to $\perp$ in case of an error or an abort).
- Reveal(P,i) $\rightarrow \{\mathsf{K}, \perp\}$: if the party has not terminated in an accepting state, this oracle outputs $\perp$; else, it outputs the session keys computed by instance P_i.
- Corrupt(P) $\rightarrow \mathsf{sk}_{\mathsf{P}}$: if P is a client, this oracle returns the long-term client key sk_{P}, but not $\mathsf{sk}_{\mathsf{Op}}$ (in this we keep faithful to the implementation of the protocol, which protects the key even from the user himself). If P is corrupted, then this party (and all its instances, past, present, or future), are considered to be adversarially controlled. If P is a server, then this oracle returns the identifier S_i, giving the adversary access to a special oracle $\mathsf{OpAccess}$.
- OpAccess(S, C) $\rightarrow m$: for a corrupted server S, this oracle gives the adversary one access to the server's local copy of all the operators, in particular returning the message that the operator Op would have output to the server on input a client C.
- StReveal(C, i, $\mathsf{bit}_{\mathsf{S}}$) $\rightarrow x$: for a client P, if $\mathsf{bit}_{\mathsf{S}} = 0$, then this oracle reveals the current state of C_i; else, if $\mathsf{bit}_{\mathsf{S}} = 1$, then the oracle returns the state the operator stores for C.
- $\mathsf{Test}^{\mathsf{K.Sec}}$($\mathsf{P}$,$i$) $\rightarrow \hat{\mathsf{K}}$: this oracle is initialized with a secret random bit b. It returns $\perp$ if the instance P_i is unfresh or if it has not terminated in an accepting state (with a session key K). If $b = 0$, then the oracle returns $\hat{\mathsf{K}} := \mathsf{K}$, else it

returns $\hat{\mathsf{K}} := \mathsf{K}'$, which is a value drawn uniformly at random from the same space as K. We assume that the adversary makes a single $\mathsf{Test}^{\mathsf{K.Sec}}$ query (a standard hybrid argument can extend the notion to multiple queries). We may assume that the adversary makes only a single $\mathsf{Test}^{\mathsf{K.Sec}}()$ query since we can extend our model to the multi-query scenario by a standard hybrid argument.

We allow the adversaries to learn whether instances have terminated and whether they have accepted or rejected their partners. Indeed, the adversary can always use Send queries to verify the status of a session. Though we do not model the precise error messages received by the two parties on abort, this seems to have no effect on the key-indistinguishability and impersonation properties of the two parties respectively. We also assume that the adversary will learn the session and partner identifiers for any session in which the instance has terminated in an accepting state.

Correctness and Partners. Each instance of each party keeps track of a session ID string, denoted sid. For the AKA protocol, this value consists of a triple of values: a user ID UID (corresponding to a single client C), a session-specific random value, and the sequence number used for the authentication step. We describe this in more detail in Section 4. We define partners as party instances that share the same session ID. More formally:

Definition 1. ***[Partners.]*** *Two instances* P_i *and* P'_j *are* partnered *if the following statements hold:*

(i) One of the parties is a user and the other is the server.
(ii) The two instances terminate in an accepting state.
(iii) The instances share the same sid*.*

In this case, the partner ID of some party P *denotes its (intended) partner.*

We define the correctness of the protocol as follows.

Definition 2. ***[Correctness.]*** *An execution of the protocol* Π *between two instances is correct if the execution is untampered with and if the following conditions hold:*

(i) The two conversing instances share the same sid*, i.e. they are partnered.*
(ii) The both instances output the same session key(s) K*.*
(iii) The partner identifiers pid *of the instances are correct, i.e they corresponds to the both conversing entities.*

We consider two classes of adversaries, *weak* and *strong*, depending on whether the adversary may corrupt servers or not. We model three requirements with respect to MiM adversaries.

Key-indistinguishability. For the property of key-indistinguishability, i.e. the guarantee that the session keys of honest sessions are indistinguishable from random, we could consider two types of models. The simpler of these gives the

adversary the ability of recovering the secret key of the operator, which considerably eases the simulation in our proof. However, we note that the operator keys are not easy to recovery by a client in real-world implementations, as they are never stored on the SIM card[6]. Thus, a more realistic model is the one we present above, in which only the client key is recovered upon corruption. We give the alternative security model in the Appendix.

The key-indistinguishability game is played as follows. First the challenger generates the keys of all the n_{Op} operators and gives black-box access to the server S. The adversary is then allowed to query any of the oracles above. We implicitly assume that the $\mathsf{Test}^{\mathsf{K.Sec}}$ oracle keeps state and, once it is queried a first time, it will return $\perp$ on all subsequent queries (we only allow a single query). However, we do allow the adversary to interact with other oracles after the $\mathsf{Test}^{\mathsf{K.Sec}}$ query as well.

Eventually, the adversary $\mathcal{A}$ outputs a bit d, which is a guess for the bit b used internally in the $\mathsf{Test}^{\mathsf{K.Sec}}$ oracle. The adversary *wins* if and only if: $b = d$ and $\mathcal{A}$ has queried a fresh instance to the $\mathsf{Test}^{\mathsf{K.Sec}}$ oracle. We consider the following definition of a fresh instance for the key-indistinguishability. We note that this notion is classical in symmetric-key protocols.

Definition 3. ***[Freshness: Key-indistinguishability.]*** *An instance P_i is* fresh *if neither this instance, nor a partner of P_i is adversarially-controlled, i.e has not been corrupted, and the following queries were not previously executed:*

(i) ***Reveal(.)****, either on the instance P_i, or on of its partners.*
(ii) ***Corrupt(.)*** *on any instance, either of P, or of their partners.*

The advantage of $\mathcal{A}$ in winning the key-indistinguishability game is defined as:

$$\mathsf{Adv}^{\mathsf{K.Ind}}_{\Pi}(\mathcal{A}) := |\Pr[\mathcal{A}\ \mathsf{wins}] - 1/2|.$$

We quantify the adversary's maximal advantage as a function of her resources which are the running time t, the number q_{exec} of instantiated party instances, and the maximum number of allowed resynchronization attempts q_{res} per instantiated instance.

Definition 4. [**Weak/Strong Key-Indistinguishability.**] *A key-agreement protocol Π is $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{G}}, \epsilon)$-weakly key-indistinguishable (resp. $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{s}}, q_{\mathsf{Op}}, q_{\mathsf{G}}, \epsilon)$-strongly-key-indistinguishable) if no adversary running in time t, creating at most q_{exec} party instances with at most q_{res} resynchronizations per instance, (corrupting at most q_{s} servers and making at most q_{Op} $\mathsf{OpAccess}$ queries per operator per corrupted server for strong security), and making at most q_{G} queries to function G, has an advantage $\mathsf{Adv}^{\mathsf{K.Ind}}_{\Pi}(\mathcal{A}) > \epsilon$.*

[6] Instead, what *is* stored in the SIM card is an intermediate value, obtained after a first Keccak truncated permutation; thus the operator key is easy to use, but hard to recover.

Client impersonation resistance. Though the AKA protocol claims to provide mutual authentication, its design introduces a vulnerability, leading to a subtle difference between the degree of *client*-impersonation resistance and *server*-impersonation resistance. In fact, as detailed in the paragraph below, the protocol allows the adversary to do a type of Man-in-the-Middle attack which resembles, but is not quite the same as, a relay attack.

We have two choices in modeling the client and server impersonation guarantees. The classical Bellare-Rogaway model, using the notion of freshness, cannot differentiate well between client- and server-impersonation resistance. A consequence is that we would only be able to prove a weaker client-impersonation guarantee than the one provided by the protocol. In our full version we also outline these notions and the respective proofs, see the Appendix.

The alternative is to give a more acurate model, which features time and can capture the difference between online and offline relays. This is the strategy we use here. In a style akin to the distance-bounding model of Dürholz et al. [24], we introduce a time variable with positive integer values, denoted clock, which increments by 1 both when a Send query is sent by the adversary, and when an honest party responds to this query. Running the Execute query increments clock by 1 for each implicit Send and for each implicit response step. For client impersonation, the only attacks we rule out are online relay attacks, which are (somewhat simplistically) depicted in Figure 2. In particular, we need to propose a more subtle definition of a fresh instance as follows:

Definition 5. *[**Freshness:** C.Imp **resistance.**] An instance S_i, with session ID* sid *and partner ID* pid*, is* fresh *if: neither this instance nor a partner of S_i is adversarially-controlled; and there exists no instance C_j sharing session* sid *with the partner* $\mathsf{pid} = \mathsf{S}_i$ *(the related transcript is denoted as* $(\mathsf{m}, \mathsf{m}', \mathsf{m}'')$*) such that the following events occur::*

(i) The message m *is sent by the adversary $\mathcal{A}$ to S_i via a* $\mathsf{Send}(m)$ *query at time* $\mathsf{clock} = k$*, yielding message m' at time* $\mathsf{clock} = k + 1$*.*
(ii) The message m′ *is sent by $\mathcal{A}$ to C_j via a* $\mathsf{Send}(m')$ *query at time* $\mathsf{clock} = k' > k + 1$*, yielding message m'' at time* $\mathsf{clock} = k' + 1$*.*
(iii) The message m″ *is sent by $\mathcal{A}$ to S_i via a* $\mathsf{Send}(m'')$ *query at time* $\mathsf{clock} = k'' > k' + 1$*.*

We note that the messages need not be exactly sequential (i.e. the adversary could query other oracles in different sessions before returning to session sid). Furthermore, the notion of freshness only refers to relays with respect to the partner client pid. We do not restrict the adversary from forwarding received messages to other server or client instances.

The goal of a client-impersonation adversary is to make a fresh server instance terminate in an accepting state. In this case, the Test oracle is not used. More formally, the game begins by generating the operator keys as before; then the adversary $\mathcal{A}$ gains access to all the oracles except $\mathsf{Test}^{\mathsf{K.Sec}}$. When $\mathcal{A}$ stops, she *wins* if there exists an instance S_i that ends in an accepting state and is fresh

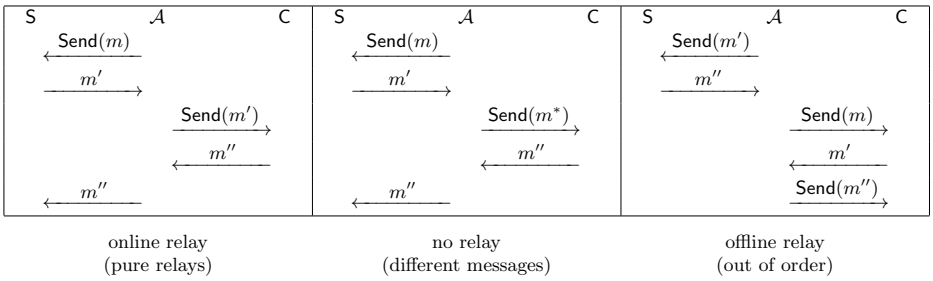


Fig. 2. Examples of Online and Offline relays.

as described above. The advantage of the adversary is defined as her success probability, i.e.

$$\mathsf{Adv}^{\mathsf{C.Imp}}_{\Pi}(\mathcal{A}) := \Pr[\mathcal{A} \text{ wins}].$$

Definition 6. [**Weak/Strong Client-Impersonation security.**]
A key-agreement protocol Π *is* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{G}}, \epsilon)$*-weak-client-impersonation-secure (resp.* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{s}}, q_{\mathsf{Op}}, q_{\mathsf{G}}, \epsilon)$*-strong-client-impersonation secure) if no adversary running in time* t*, creating at most* q_{exec} *party instances with at most* q_{res} *resynchronizations per instance, (corrupting at most* q_{s} *servers and making at most* q_{Op} OpAccess *queries per operator per corrupted server for strong security), and making at most* q_{G} *queries to the function* G*, has an advantage* $\mathsf{Adv}^{\mathsf{C.Imp}}_{\Pi}(\mathcal{A}) \geq \epsilon$*.*

Server impersonation resistance. As we explain in more detail in Section 6, it is possible to impersonate a server even if we rule out online relays. In particular, an adversary performs an offline (out of order) relay, as described in the third scenario of Figure 2. This is because the client's first message is the user id, which is always sent in clear (thus known to adversaries). This enables $\mathcal{A}$ to obtain, in a first session with the server, the server's authenticated challenge for a particular client UID, which it can replay to UID, in a separate (later) session. In essence, the adversary is relaying the messages, but this happens in two different, non-concurrent executions. This indicates a gap between the client impersonation and the server impersonation guarantees for the AKA protocol.

Our server-impersonation model rules out both offline and online relays, redefining freshness as follows:

Definition 7. ***[Freshness:*** S.Imp ***resistance.]*** *An instance* C_i*, with session ID* sid *and partner ID* pid*, is* fresh *if: neither this instance nor a partner of* C_i *is adversarially-controlled; and there exists no instance* S_j *with session* sid *and partner* $\mathsf{pid} = \mathsf{C}_i$ *(the transcript of* sid *is denoted as* $(\mathsf{m}, \mathsf{m}', \mathsf{m}'')$*) such that the following events occur:*

(i) The message m *is sent by* $\mathcal{A}$ *to* S_j *via a* Send(m) *query yielding message* m'*.*

(ii) The message m' *is sent by* $\mathcal{A}$ *to* C_i *via a* $\mathsf{Send}(m')$ *query yielding message* m''.
(iii) The message m'' *is sent by* $\mathcal{A}$ *to* S_j *via a* $\mathsf{Send}(m'')$ *query.*

The game is played as in the client impersonation case. When the adversary $\mathcal{A}$ stops, she *wins* if there exists a fresh instance C_i that ends in an accepting state. The advantage of the adversary is defined as its success probability, i.e.

$$\mathsf{Adv}_{\Pi}^{\mathsf{S.Imp}}(\mathcal{A}) := \Pr[\mathcal{A} \text{ wins}].$$

Definition 8. [**Weak/Strong Server-Impersonation security.**]
A key-agreement protocol Π *is* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{G}}, \epsilon)$*-weak-server-impersonation-secure (resp.* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{s}}, q_{\mathsf{Op}}, q_{\mathsf{G}}, \epsilon)$*-strong-server-impersonation secure) if no adversary running in time* t*, creating at most* q_{exec} *party instances with at most* q_{res} *resynchronizations per instance, (corrupting at most* q_{s} *servers and making at most* q_{Op} $\mathsf{OpAccess}$ *queries per operator per corrupted server for strong security), and making at most* q_{G} *queries to the function* G*, has an advantage* $\mathsf{Adv}_{\Pi}^{\mathsf{S.Imp}}(\mathcal{A}) \geq \epsilon$.

3.3 Security w.r.t servers.

In this section, we consider a new adversary where she is a malicious but legitimate server S. Indeed, a such context requires that (malicious) server cannot learn any secret data of the subscriber or operator, i.e the subscriber key sk_{C}, operator key $\mathsf{sk}_{\mathsf{Op}}$ and the two related internal states. Moreover, the server must not be able to make a client accept the server's authentication (thus completing the key-derivation process), unless they are explicitly given authenticating information by a legitimate operator. We formalize these goals in terms of two new properties: key-confidentiality and soundness. This model is really similar as the previous one, and is based on the same participants which includes the adversary. For both properties, the adversary uses the UReg, **NewInstance**, $\mathsf{Execute}$, Send, Reveal, $\mathsf{StReveal}$ oracles as described in the previous model. We additionally add two new oracles (including a new $\mathsf{Corrupt}$ oracle) as noted below:

- $\mathsf{Corrupt}(\mathsf{P}) \rightarrow S$: if P is a client, behave as before in the previous model. If P is an operator, returns $\mathsf{sk}_{\mathsf{Op}}$ and the list of tuples $S = (\mathsf{UID}, \mathsf{sk}_{\mathsf{UID}}, \mathsf{st}_{\mathsf{UID}}, \mathsf{st}_{\mathsf{Op,C}})$ for all clients C subscribing with that operator.
- $\mathsf{OpAccess}(\mathsf{C}) \rightarrow m$: this oracle gives the adversary one access to the server's local copy of all the operators, in particular returning the message m that the operator Op would have output to the server on input a client C.

Unlike key-indistinguishability, which guarantees that *session* keys are indistinguishable from random with respect to *MiM* adversaries, the property of key confidentiality demands that *long-term* client keys remain confidential with respect to *malicious servers*

This game begins by generating the material for n_{Op} operators and n_{C} clients. The adversary can then interact arbitrarily with these entities by using the

oracles above. At the end of the game, the adversary must output a tuple: $(\mathsf{P}_i, \mathsf{sk}^*_{\mathsf{UID}}, \mathsf{sk}^*_{\mathsf{Op}}, \mathsf{st}^*_{\mathsf{UID}}, \mathsf{st}^*_{\mathsf{Op,UID}})$ such that UID is the long-term identifier of P and P_i is a fresh instance of P in the sense formalized below. The adversary wins if at least one of the values: $\mathsf{sk}^*_{\mathsf{UID}}, \mathsf{sk}^*_{\mathsf{Op}}, \mathsf{st}^*_{\mathsf{UID}}, \mathsf{st}^*_{\mathsf{Op,UID}}$ is respectively equal to $\mathsf{sk}_{\mathsf{UID}}, \mathsf{sk}_{\mathsf{Op}}, \mathsf{st}_{\mathsf{UID}}, \mathsf{st}_{\mathsf{Op,UID}}$, the real secret values of the fresh instance P_i.

Definition 9. *[Freshness:* $\mathsf{St.Conf}$*] An instance* P_i *is* fresh *if neither this instance, nor a partner of* P_i *is adversarially-controlled (its long-term key* sk_{P} *has not been corrupted) and the following queries were not previously executed:*

(i) $\mathsf{StReveal}(.)$ *on* any *instance of* P*.*
(ii) $\mathsf{Corrupt}(.)$ *on* any *instance of* P *or on the operator* Op *to which* P *subscribes.*

The advantage of the adversary is defined as:

$$\mathsf{Adv}^{\mathsf{St.Conf}}_{\Pi}(\mathcal{A}) := \Pr[\mathcal{A} \text{ wins}].$$

Definition 10. [State-confidentiality.] *A key-agreement protocol* Π *is* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{Op}}, q_{\mathsf{G}}, \epsilon)$*-state-confidential if no adversary running in time* t*, creating at most* q_{exec} *party instances with at most* q_{res} *resynchronizations per instance, making at most* q_{Op} $\mathsf{OpAccess}$ *queries and* q_{G} *queries to* G*, has an advantage* $\mathsf{Adv}^{\mathsf{St.Conf}}_{\Pi}(\mathcal{A}) \geq \epsilon$*.*

In the *Soundness* game, we demand that no server is able to make a fresh client instance terminate in an accepting state without help from the operator. This game resembles impersonation-security; however, this time the adversary is a legitimate server (not a MiM) and it has access to operators. The adversary may interact with oracles in the soundness game arbitrarily, but we only allow a maximum number of q_{Op} queries to the $\mathsf{OpAccess}$ oracle per client.

The adversary wins if there exist $(q_{\mathsf{Op}} + 1)$ fresh client instances of a given client which terminated in an accepting state. Freshness is defined similarly as in the impersonation game with the same restriction due to the offline replays attacks:

Definition 11. ***[Freshness: soundness resistance.]*** *An instance* C_i*, with session ID* sid *and partner ID* pid*, is* fresh *if: neither this instance, a partner of* C_i *nor their related operator* Op *is adversarially-controlled ;and there exists no instance* S_j *with session* sid *and partner* $\mathsf{pid} = \mathsf{C}_i$ *(the transcript of* sid *is denoted as* $(\mathsf{m}, \mathsf{m}', \mathsf{m}'')$*) such that the following events occur:*

(i) The message m *is sent by* $\mathcal{A}$ *to* S_j *via a* $\mathsf{Send}(m)$ *query yielding message* m'*.*
(ii) The message m' *is sent by* $\mathcal{A}$ *to* C_i *via a* $\mathsf{Send}(m')$ *query yielding message* m''*.*
(iii) The message m'' *is sent by* $\mathcal{A}$ *to* S_j *via a* $\mathsf{Send}(m'')$ *query.*

The advantage of the adversary is defined as:

$$\mathsf{Adv}_{\Pi}^{\mathsf{Sound}}(\mathcal{A}) := \Pr[\mathcal{A} \text{ wins}].$$

Definition 12. [Soundness.] *A key-agreement protocol* Π *is* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{Op}}, q_{\mathsf{G}}, \epsilon)$*-server-sound if no adversary running in time* t*, creating at most* q_{exec} *party instances with at most* q_{res} *resynchronizations per instance, making at most* q_{Op} *queries to any operator* Op *and at most* q_{G} *queries to the function* G*, has an advantage* $\mathsf{Adv}_{\Pi}^{\mathsf{Sound}}(\mathcal{A}) \geq \epsilon$*.*

4 Security of the AKA protocol

In this section, we focus on the *current, unmodified* version of the AKA protocol with respect to the five properties formalized in Section 3.

In particular, parties P (clients C and servers S) run sessions of the protocol, thus creating party *instances* denoted P_i. An instance is said to finish in an *accepting* state if and only if it auhenticates its partner. Each instance keeps track of a partner- and a session-ID.

The partner ID pid of an accepting client instance C_i is S (this reflects the lack of server identifiers); server instances S_i, have a pid corresponding to a unique UID. The session ID sid of each instance consists of: UID, R, and the value Sqn that is agreed upon during the session. In the absence of resynchronization, the session ID is $(\mathsf{UID}, \mathsf{R}, \mathsf{Sqn}_{\mathsf{Op},\mathsf{C}})$. During re-synchronization, the operator updates $\mathsf{Sqn}_{\mathsf{Op},\mathsf{C}}$ to the client's $\mathsf{Sqn}_{\mathsf{C}}$; this update is taken into account in the sid. Any two partners (same sid) with accepting states compute session keys $(\mathsf{CK}\|\mathsf{IK})$.

A Unitary Function G**.** We analyse the security of AKA in two steps. First, we reduce the security of AKA to the security (pseudorandomness) of an intermediate, unitary function G. This function models the suite of seven algorithms used in AKA; each algorithm is a specific call to G. For the state-confidentiality property we also need to assume the pseudorandomness of the related unitary function G^*, which is the same as G, but we key it with the operator key $\mathsf{sk}_{\mathsf{Op}}$ rather than the client key sk. Thus, this first step gives a sufficient condition to provide AKA security for any suite of algorithms intended to be used within it. As a second step (showed in the full version), we prove that both current proposals for AKA, i.e. TUAK and MILENAGE, guarantee this property.

We note that the pseudorandomness of the unitary function G implies the pseudorandomness of each of the sub-algorithms; however, it is a strictly stronger property, which is necessary because e.g. the session keys CK and IK, computed by two different algorithms on the same input, *must* be independent.

4.1 Provable Security Guarantees

The existing AKA protocol only attains the weaker versions of key-indistinguishability, client-, and server-impersonation resistance. The protocol also guarantees state-confidentiality and soundness with respect to malicious servers.

Denote by Π the AKA protocol described in Section 2.2, but in which the calls to the internal cryptographic functions $\mathcal{F}_1, \ldots, \mathcal{F}_5, \mathcal{F}_1^*, \mathcal{F}_5^*$ are replaced by calls to the function $G : \{0,1\}^\kappa \times \{0,1\}^d \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^n$, in which κ is a security parameter, d is a positive integer strictly larger than the size of the operator key, and t indicates the block size of an underlying pseudo-random permutation. Each input space is specified according the considered instantiations (MILENAGE or TUAK)detailed above.

We denote by $\mathcal{S}_\mathsf{C} := \{0,1\}^\kappa$ the key-space for the client keys and by $\mathcal{S}_{\mathsf{Op}} := \{0,1\}^e$, the key space for operator keys, for some specified $e < d$ (in practice $e = 256$). Our system features n_C clients, $\mathsf{n_S}$ servers and n_{Op} operators.

MILENAGE and TUAK as G. In appendix F, we prove that both the calls to the instantiations (see Appendix E) MILENAGE and TUAK algorithms can be modeled as the unitary function G that we use for our proofs. The last step in our proof is to show that both algorithm suites exhibit the PRF property we require for G, when instantiated with the key $\mathsf{sk_C}$ **and** with the operator key $\mathsf{sk_{Op}}$ (for the key-confidentiality). However, as opposed to TUAK (whose symmetric design allows a lot more leeway), the MILENAGE algorithms require a stronger assumption to prove the PRF property when G is used with key $\mathsf{sk_{Op}}$ (see Appendix F)

Security statements.

We proceed to give the five security statements with the respect to the AKA protocol, in the following order: first, the notion of weak-key-secrecy, then strong-client-, and weak-server-impersonation resistance, soundness with respect to servers, and finally the state-confidentiality, which requires an additional assumption. We include proofs for these properties in Appendix D.

Theorem 1. *[W.K.Ind-resistance.] Let $G : \{0,1\}^\kappa \times \{0,1\}^d \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^n$ be our specified function specified in section 4 and Π the AKA protocol specified in section 4. Consider a $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_G)$-adversary $\mathcal{A}$ against the W.K.Ind-security of the protocol Π, running in time t and creating at most q_{exec} party instances with at most q_{res} resynchronizations per instance, and making q_G queries to the function G. Denote the advantage of this adversary as $\mathsf{Adv}_\Pi^{\mathsf{W.K.Ind}}(\mathcal{A})$. Then there exists a $(t' \approx O(t), q' = q_G + q_{\mathsf{exec}}(2 + q_{\mathsf{res}}))$-prf-adversary $\mathcal{A}'$ on G such that:*

$$\mathsf{Adv}_\Pi^{\mathsf{W.K.Ind}}(\mathcal{A}) \leq n_\mathsf{C} \cdot \left(\frac{q_{\mathsf{exec}}^2}{2^{|\mathsf{R}|}} + \mathsf{Adv}_G^{\mathsf{prf}}(\mathcal{A}') \right).$$

Thus, we show that this result holds (in fact with a simpler proof) even if operator key corruptions are possible. This is an important, as it shows that even if an adversary knows all the operator keys, it is still unable to distinguish real session keys from random ones. This is also the case for the client- and server-impersonation statements below.

Theorem 2. *[S.C.Imp-resistance.] Let $G : \{0,1\}^\kappa \times \{0,1\}^d \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^n$ be our specified function specified in section 4 and Π the AKA protocol*

specified in section 4. Consider a $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{s}}, q_{\mathsf{Op}}, q_G)$*-adversary* $\mathcal{A}$ *against the* $\mathsf{S.C.Imp}$*-security of the protocol* Π*, running in time* t *and creating at most* q_{exec} *party instances with at most* q_{res} *resynchronizations per instance, corrupting at most* q_{s} *servers and making at most* q_{Op} $\mathsf{OpAccess}$ *queries per operator per corrupted server and making* q_G *queries to the function* G*. Denote the advantage of this adversary as* $\mathsf{Adv}_{\Pi}^{\mathsf{S.C.Imp}}(\mathcal{A})$*. Then there exists a* $(t' \approx O(t), q' = 5 \cdot q_{\mathsf{Op}} \cdot q_{\mathsf{s}} + q_G + q_{\mathsf{exec}}(q_{\mathsf{res}} + 2))$*-prf-adversary* $\mathcal{A}'$ *on* G *such that:*

$$\mathsf{Adv}_{\Pi}^{\mathsf{S.C.Imp}}(\mathcal{A}_{\mathbb{G}_0}) \leq n_{\mathsf{C}} \cdot \left(2 \cdot \mathsf{Adv}_{G}^{\mathsf{prf}}(\mathcal{A}') + \frac{(q_{\mathsf{exec}} + q_{\mathsf{s}} \cdot q_{\mathsf{Op}})^2}{2^{|\mathsf{R}|}} + \frac{q_{\mathsf{exec}} \cdot q_{\mathsf{res}}}{2^{|\mathsf{Res}|}} + \frac{1}{2^{\kappa}}\right).$$

Theorem 3. *[*$\mathsf{W.S.Imp}$***-resistance.]** *Let* $G : \{0,1\}^{\kappa} \times \{0,1\}^{d} \times \{0,1\}^{t} \times \{0,1\}^{t} \rightarrow \{0,1\}^{n}$ *be our specified function specified in section 4 and* Π *our protocol specified in section 4. Consider a* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{G}})$*-adversary* $\mathcal{A}$ *against the* $\mathsf{W.S.Imp}$*-security of the protocol* Π*, running in time* t*, creating at most* q_{exec} *party instances, running at most* q_{res} *re-synchronizations per each instance, and making at most* q_{G} *queries to the function* G*. Denote the advantage of this adversary as* $\mathsf{Adv}_{\Pi}^{\mathsf{W.S.Imp}}(\mathcal{A})$*. Then there exists a* $(t' \approx t, q = q_{\mathsf{exec}} \cdot (q_{\mathsf{res}} + 2) + q_{\mathsf{G}})$*-adversary* $\mathcal{A}$*'* *with an advantage* $\mathsf{Adv}_{G}^{\mathsf{prf}}(\mathcal{A}')$ *of winning against the pseudorandomness of the function* G*, such that:*

$$\mathsf{Adv}_{\Pi}^{\mathsf{W.S.Imp}}(\mathcal{A}) \leq n_{\mathsf{C}} \cdot \left(\mathsf{Adv}_{G}^{\mathsf{prf}}(\mathcal{A}') + \frac{q_{\mathsf{exec}} \cdot q_{\mathsf{res}}}{2^{|\mathsf{Mac_S}|}} + \frac{1}{2^{\kappa}}\right).$$

Theorem 4. *[*Sound***-security.]** *Let* $G : \{0,1\}^{\kappa} \times \{0,1\}^{d} \times \{0,1\}^{t} \times \{0,1\}^{t} \rightarrow \{0,1\}^{n}$ *be our specified function in section 4 and* Π *the protocol specified in section 4. Consider a* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{Op}}, q_{\mathsf{G}}, \epsilon)$*-server-sound-adversary* $\mathcal{A}$ *against the soundness of the protocol* Π*, running in time* t*, creating at most* q_{exec} *party instances with at most* q_{res} *resynchronizations per instance, making at most* q_{Op} *queries to any operator* Op *and at most* q_{G} *queries to the function* G*. Denote the advantage of this adversary as* $\mathsf{Adv}_{\Pi}^{\mathsf{Sound}}(\mathcal{A})$*. Then there exists a* $(t' \approx t, q' = 5 \cdot q_{\mathsf{Op}} + q_G + q_{\mathsf{exec}}(2 + q_{\mathsf{res}}))$*-adversary* $\mathcal{A}'$ *with an advantage* $\mathsf{Adv}_{G}^{\mathsf{prf}}(\mathcal{A}')$ *of winning against the pseudorandomness of the function* G*, such that:*

$$\mathsf{Adv}_{\Pi}^{\mathsf{Sound}}(\mathcal{A}) \leq n_{\mathsf{C}} \cdot \left(2 \cdot \mathsf{Adv}_{G}^{\mathsf{prf}}(\mathcal{A}') + \frac{q_{\mathsf{exec}} \cdot q_{\mathsf{res}}}{2^{|\mathsf{Mac_S}|}} + \frac{1}{2^{\kappa}}\right).$$

Theorem 5. *[*$\mathsf{St.Conf}$***-resistance.]** *Let* G *and* G^* *be our specified functions specified in section 4 and* Π *our fixed variant of the* AKA *protocol specified in section 4. Consider a* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{Op}}, q_G, q_{G^*})$*-adversary* $\mathcal{A}$ *against the* $\mathsf{St.Conf}$*-security of the protocol* Π*, running in time* t *and creating at most* q_{exec} *party instances with at most* q_{res} *resynchronizations per instance, making at most* q_{Op} *queries to oracle* $\mathsf{OpAccess}$ *and making* q_G *(resp.* q_{G^*}*) queries to the function* G *(resp.* G^**). Denote the advantage of this adversary as* $\mathsf{Adv}_{\Pi}^{\mathsf{St.Conf}}(\mathcal{A})$*. Then there exist a* $(t' \approx O(t), q' = q_G + q_{\mathsf{exec}}(5 + q_{\mathsf{res}}))$*-prf-adversary* $\mathcal{A}_1$ *on* G *and*

$(t' \approx O(t), q' = q_{G^*})$-*prf-adversary* $\mathcal{A}_2$ *on* G^* *a such that:*

$$\mathsf{Adv}^{\mathsf{St.Conf}}_{\Pi}(\mathcal{A}) \leq n_{\mathsf{C}} \cdot \left(\frac{1}{2^{|\mathsf{sk_C}|}} + \frac{1}{2^{|\mathsf{sk_{Op}}|}} + \frac{2}{2^{|\mathsf{Sqn}|}} + \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}_1) + \mathsf{Adv}^{\mathsf{prf}}_{G^*}(\mathcal{A}_2) \right).$$

4.2 Vulnerabilities of the AKA protocol

In the three-party mobile setting, the server is authenticated by the client if it presents credentials (authentication vectors) generated by the client's operator. The properties of state-confidentiality and soundness, which the AKA protocol guarantees, indicate that servers cannot learn the client's long-term data, and that they cannot authenticate without the operator-generated data.

However, Zhang [18] and Zhang and Fang [19] pointed out that once a server is corrupted, it can obtain legitimate authentication data from the client's operator, and then use this data to set up a False Base Station (FBS), which can lead to a malicious, unauthorised server authenticating to the client. As a result, the AKA protocol does not guarantee strong key-indistinguishability, nor strong server-impersonation resistance.

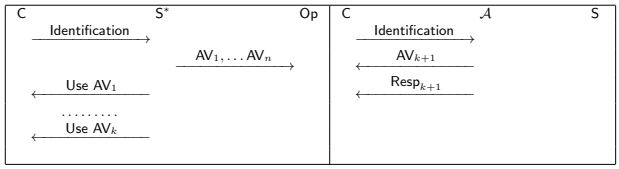


Fig. 3. The attack of Zhang and Fang. On the left hand side, the client is in the vulnerable network, interacting with the server S^*. The server uses up authentication vectors $\mathsf{AV}_1, \ldots \mathsf{AK}_k$. Then, the server S^* is corrupted, and the adversary $\mathcal{A}$ learns $\mathsf{AV}_{k+1}, \ldots, \mathsf{AV}_n$, which it uses in a second attack phase (on the right).

The main attack strategy is also depicted in Figure 3. In a first step, the client C is assumed to be in the LAI corresponding to a server S^*, which will later be corrupted. The server receives a batch of authentication vectors $(\mathsf{AV}_1, \ldots, \mathsf{AV}_n)$, using some of them (vectors $\mathsf{AV}_1, \ldots, \mathsf{AV}_k$) to provide service to that client (and learn what services this client has provided, etc.). Subsequently, the client moves to a different LAI, outside the corrupted network's area. The adversary $\mathcal{A}$ has corrupted the server S^* and learned the remaining vectors $\mathsf{AV}_{k+1}, \ldots, \mathsf{AV}_n$; this adversary then uses this authentication data to authenticate to the client, *in its new location*. This immediately breaks the server-impersonation guarantee. Moreover, since authentication vectors also contain the short-term session keys, key-indistinguishability is breached, too. This attack is particularly dangerous since a single server corruption can affect a very large number of clients. Moreover, server corruption is easily practiced in totalitarian regimes, in which mobile

providers are subject to the state, and partial data is furthermore likely to be leaked upon using backdoored algorithms.

This attack does not, however, affect client-impersonation resistance, since the server cannot use an authentication vector from the server to respond to a freshly-generated authentication challenge (the random value for the two authentication vectors is different).

5 Additional Security with few modifications

The main reason server-corruption attacks are effective is that servers associated with a specific geographic area (like a country, a region, etc.) can re-use authentication vectors given by the operator in a different geographic area, impersonating the legitimate server associated with that area. This vulnerability, however, is easily fixed as long as the client's device is aware of its geographical location. Our solution is to add a unique server identifier, denoted $\mathsf{Id_S}$, to the input of each of the cryptographic functions, thus making any leftover authentication tokens un-replayable in the wrong area. We stress that this is a minor modification to the protocol, as servers are already associated with a unique LAI identifier. We also show in Appendix E how to include $\mathsf{Id_S}$ in the computation of each of the cryptographic algorithms. We present our modified protocol in Figure 4.

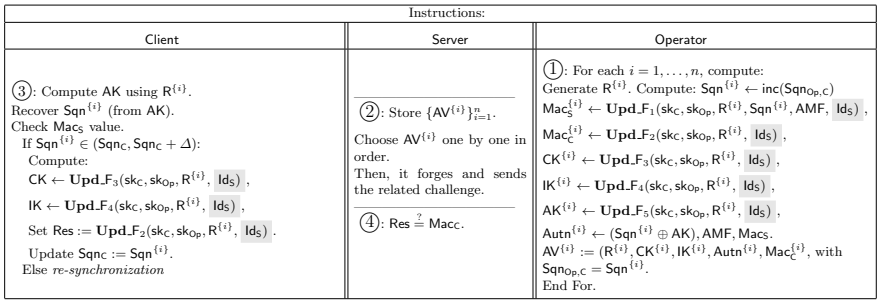


Fig. 4. The modified instructions of the fixed AKA Procedure.

Security of the modified AKA protocol. This modification still (trivially) preserves the properties of strong client-impersonation resistance, soundness, and state confidentiality. However, the modification yields in addition strong key-indistinguishability and server-impersonation resistance, as we detail below. The proofs are detailed in Appendix D.

Theorem 6. *[$\mathsf{S.K.Ind}$-resistance.] Let $G : \{0,1\}^\kappa \times \{0,1\}^d \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^n$ be our specified function specified in section 4 and Π the fixed AKA protocol specified in section 4. Consider a $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{s}}, q_{\mathsf{Op}}, q_G)$-adversary $\mathcal{A}$ against the $\mathsf{S.K.Ind}$-security of the protocol Π, running in time t and creating at most*

q_{exec} *party instances with at most* q_{res} *resynchronizations per instance, corrupting at most* q_{s} *servers, making at most* q_{Op} *queries per operator per corrupted server and making* q_G *queries to the function* G*. Denote the advantage of this adversary as* $\mathsf{Adv}^{\mathsf{S.K.Ind}}_{\Pi}(\mathcal{A})$*. Then there exists a* $(t' \approx O(t), q' = 5 \cdot q_{\mathsf{Op}} + q_G + q_{\mathsf{exec}}(q_{\mathsf{res}} + 2))$*-prf-adversary* $\mathcal{A}'$ *on* G *such that:*

$$\mathsf{Adv}^{\mathsf{S.K.Ind}}_{\Pi}(\mathcal{A}) \leq n_{\mathsf{C}} \cdot \left(\frac{(q_{\mathsf{exec}} + q_{\mathsf{s}} \cdot q_{\mathsf{Op}})^2}{2^{|\mathsf{R}|}} + 2 \cdot \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}') \right).$$

Theorem 7. *[*S.S.Imp*-resistance.] Let* $G : \{0,1\}^{\kappa} \times \{0,1\}^{d} \times \{0,1\}^{t} \times \{0,1\}^{t} \rightarrow \{0,1\}^{n}$ *be our specified function specified in section 4 and* Π *our fixed variant of the* AKA *protocol specified in section 4. Consider a* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{s}}, q_{\mathsf{Op}}, q_{\mathsf{G}}, \epsilon)$*-adversary* $\mathcal{A}$ *against the* S.S.Imp*-security of the protocol* Π*, running in time* t *and creating at most* q_{exec} *party instances with at most* q_{res} *resynchronizations per instance, corrupting at most* q_{s} *servers, making at most* q_{Op} *queries per operator per corrupted server and making* q_G *queries to the function* G*. Denote the advantage of this adversary as* $\mathsf{Adv}^{\mathsf{S.S.Imp}}_{\Pi}(\mathcal{A})$*. Then there exists a* $(t' \approx O(t), q' = 5 \cdot q_{\mathsf{s}} \cdot q_{\mathsf{Op}} + q_G + q_{\mathsf{exec}}(2 + q_{\mathsf{res}}))$*-prf-adversary* $\mathcal{A}'$ *on* G *such that:*

$$\mathsf{Adv}^{\mathsf{S.S.Imp}}_{\Pi}(\mathcal{A}_{\mathbb{G}_0}) \leq n_{\mathsf{C}} \cdot \left(\frac{q_{\mathsf{exec}} \cdot q_{\mathsf{res}}}{2^{|\mathsf{Mac_S}|}} + \frac{1}{2^{\kappa}} + 2 \cdot \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}') \right).$$

Each of the two bounds above depend linearly on the number of clients n_{C}; while this number can be as large as, potentially, six billion, the size of the secret keys (128 or 256 bits) and of the random value (128 bits) can still make the bound negligible. The linear factor n_{C}, however, highlights the importance of using authentication strings longer than 128 bits for authentication.

6 Impact

The AKA protocol was designed for 3G networks, and is currently used to securely provide service to mobile clients on 3G/4G networks (the latter is done by using AKA together with the LTE protocol). As a standardized, and highly used protocol, AKA is likely to become one of the main building blocks securing 5G communications. Despite its significance, the security of the AKA protocol is not well-understood to date. Several previous results indicate privacy flaws and propose quite radical modifications which are claimed to provide better privacy. In this paper, we have focused on the *actual security guarantees* of the unmodified AKA protocol, and we showed that a small modification which is easily incorporated in the design of AKA can provide much stronger security (i.e. with respect to corruptions).

Since it is used in 3G and 4G communications, the AKA protocol is subject to constraints dictated by its usage (in a three-party environment, rather than two parties, as usually featured in typical AKE scenarios) and by hardware restrictions (e.g. the inability of SIM cards to generate randomness). The three-party

scenario makes usual cryptographic models such as BPR hard to use, since security must also be defined with respect to the semi-trusted servers. The somewhat unorthodox and counter-intuitive design of AKA (from a cryptographic point of view) makes the analysis of its security difficult in that known results on AKE cannot be applied in a straightforward way.

Our analysis follows the design of AKA closely, and we analyse security in a strong model, which allows corruptions of both clients and servers. We show that the small modification of introducing a server-specific identifier in the cryptographic functions mitigates the consequences of server corruptions as detailed by Zhang [18]. We also show how to incorporate this modification in TUAK and MILENAGE. We prove security by relying on a classical assumption (pseudo-randomness) of a unitary function G, and we show that both the TUAK and MILENAGE algorithm-suites can be proved to behave as such a function. This gives, for any suite of algorithms with appropriate input/output domains, a *sufficient* condition to ensure the security of AKA.

A limitation of our analysis is that we consider only the security of AKA, rather than its privacy. This is partly because existing results already show that AKA does not guarantee a very strong degree of privacy. Moreover, as we show in this paper, the *security* of this protocol (with respect to key-establishment, authentication, and trust with respect to servers) is a vast topic, which has not been previous studied. An interesting direction for future research would be a thorough privacy analysis of AKA and modifying this protocol such that it is still implementable in current conditions, while providing better privacy.

References

1. 3GPP. 3G Security; Specification of the MILENAGE algorithm set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 2: Algorithm specification. TS 35.206, 3rd Generation Partnership Project (3GPP), June 2007.
2. 3GPP. 3G Security; Specification of the TUAK algorithm set: A 2nd example for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5* – Document 1: Algorithm specification. TS 35.231, 3rd Generation Partnership Project (3GPP), June 2013.
3. 3GPP. 3G Security; Technical Specification Group Services and System Aspects; 3GPP System Architecture Evolution(SAE) SA; Security Architecture. TS 33.401, 3rd Generation Partnership Project (3GPP), June 2013.
4. 3GPP. 3G Security; Technical Specification Group (TSG) SA; 3G Security; Security Architecture. TS 33.102, 3rd Generation Partnership Project (3GPP), June 2013.
5. 3GPP. 3rd Generation Partnership Project; Technical Specification Group Services ans System Aspects; Security related network functions (Release 12). TS 43.020, 3rd Generation Partnership Project (3GPP), June 2014.
6. 3GPP. 3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Numbering, addressing and identification (Release 13). TS 23.003, 3rd Generation Partnership Project (3GPP), 2016.

7. Arapinis, Myrto and Mancini, Loretta Ilaria and Ritter, Eike and Ryan, Mark and Golde, Nico and Redon, Kevin and Borgaonkar, Ravishankar. New privacy issues in mobile telephony: fix and verification. In Yu, Ting and Danezis, George and Gligor, Virgil D., editor, *ACM Conference on Computer and Communications Security*, pages 205–216. ACM, 2012.
8. Bruno Blanchet. Automatic Verification of Security Protocols in the Symbolic Model: The Verifier ProVerif. In *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, pages 54–87, 2013.
9. Chris Hall and David Wagner and John Kelsey and Bruce Schneier. Building PRFs from PRPs. In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference*, pages 370–389, 1998.
10. Guido Bertoni and Joan Daemen and Michael Peeters and Gilles Van Assche. On the Indifferentiability of the Sponge Construction. In *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques.*, pages 181–197, 2008.
11. Guido Bertoni and Joan Daemen and Michael Peeters and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In A. Miri and S. Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011.*, volume 7118 of *LNCS*, pages 320–337. Springer, 2011.
12. Guido Bertoni and Joan Daemen and Michaël Peeters and Gilles Van Assche. Keccak. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 313–314, 2013.
13. Henri Gilbert. The Security of "One-Block-to-Many" Modes of Operation. In *Fast Software Encryption, 10th International Workshop, FSE*, pages 376–395, 2003.
14. Mihir Bellare and David Pointcheval and Phillip Rogaway. Authenticated Key Exchange Secure against Dictionary Attacks. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques*, pages 139–155, 2000.
15. Mihir Bellare and Phillip Rogaway. Entity Authentication and Key Distribution. In D. R. Stinson, editor, *Advances in Cryptology -CRYPTO'93, 13th Annual International Cryptology Conference*, volume 773 of *LNCS, p.232-249*. Springer, 1994.
16. Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: the three party case. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 57–66, 1995.
17. Mihir Bellare and Ran Canetti and Hugo Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 419–428, 1998.
18. Muxiang Zhang. Provably-Secure Enhancement on 3GPP Authentication and Key Agreement Protocol. *IACR Cryptology ePrint Archive*, 2003:92, 2003.
19. Muxiang Zhang and Yuguang Fang. Security analysis and enhancements of 3GPP authentication and key agreement protocol. *IEEE Transactions on Wireless Communications*, 4(2):734–742, 2005.
20. Myrto Arapinis and Eike Ritter and Mark Dermot Ryan. StatVerif: Verification of Stateful Processes. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF*, pages 33–47, 2011.
21. Myrto Arapinis and Loretta Ilaria Mancini and Eike Ritter and Mark Ryan. Privacy through Pseudonymity in Mobile Telephony Systems. In *21st Annual Network and Distributed System Security Symposium, NDSS*, 2014.

22. Pierre-Alain Fouque and David Pointcheval and Sébastien Zimmer. HMAC is a randomness extractor and applications to TLS. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS*, pages 21–32, 2008.
23. Ran Canetti and Hugo Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In L. R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques*, volume 2332 of *LNCS*, pages 337–351, 2002.
24. Ulrich Dürholz and Marc Fischlin and Michael Kasper and Cristina Onete. A Formal Approach to Distance Bounding RFID Protocols. In *Proceedings of the* 14^{th} *Information Security Conference ISC 2011*, volume 7001 of *LNCS*, pages 47–62. Springer, 2011.
25. Victor Shoup. On Formal Models for Secure Key Exchange. *IACR Cryptology ePrint Archive*, 1999:12, 1999.

A Security notions

A.1 Security notions

The security notions can be proved under known or chosen message attacks, denoted respectively kma and cma. In this paper, we define all the security notions under the chosen messages attacks.

Pseudo-random function. A pseudo-random function (prf) is a family of functions with the property that the input-output behavior of a random instance of the family is computationally indistinguishable from that of a random function. This property is defined in terms of the following security game $\mathbb{G}^{\mathsf{prf}}$:

1. The challenger $\mathcal{C}_f^{\mathsf{prf}}$ chooses a bit $b \in \{0,1\}$. If $b = 0$, it assigns f to a random function $\mathsf{Rand} : \{0,1\}^d \to \{0,1\}^n$. Else if $b = 1$, it chooses a key $K \in \{0,1\}^\kappa$ and assigns f to the function $f(K,.)$.
2. The adversary $\mathcal{A}$ sends one by one q messages $x_i \in \{0,1\}^d$ to the challenger and receives $f(x_i)$.
3. Finally, $\mathcal{A}$ outputs a guess d of the bit b to the $\mathcal{C}_f^{\mathsf{prf}}$.

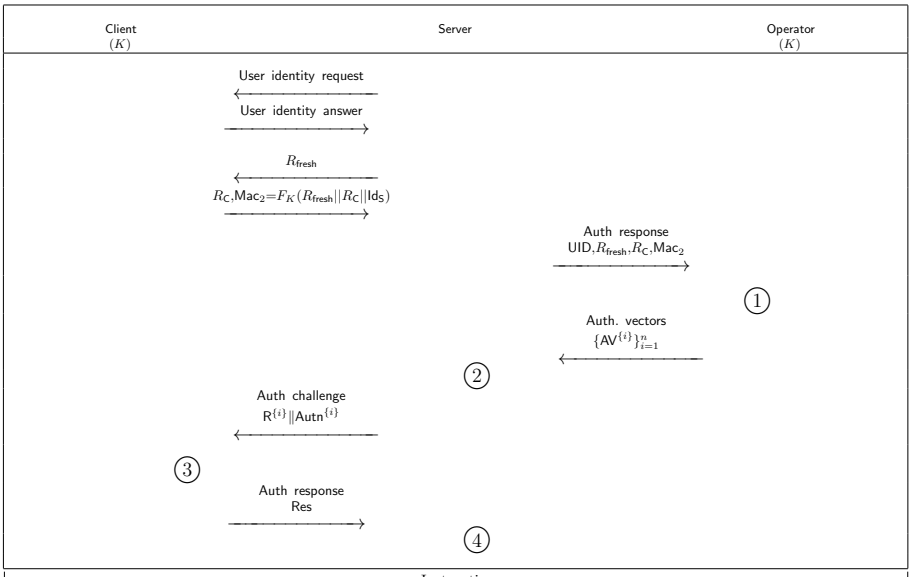
We can evaluate the prf-advantage of an adversary against f, denoted $\mathsf{Adv}_f^{\mathsf{prf}}(\mathcal{A})$ as follows, for a random function denoted $\mathsf{Rand} : \{0,1\}^d \to \{0,1\}^n$:

$$\mathsf{Adv}_f^{\mathsf{prf}}(\mathcal{A}) = \big|\Pr[\mathcal{A} \to 1 | f \xleftarrow{\$} F(K,.), K \xleftarrow{\$} \{0,1\}^\kappa] - \Pr[\mathcal{A} \to 1 | f \xleftarrow{\$} \mathsf{Rand}]\big|,$$

Definition 13. *([**Pseudo-Random Function.**]) A family f of functions from $\{0,1\}^\kappa \times \{0,1\}^d$ to $\{0,1\}^n$ is said to be (t,q)-prf-secure if any adversary $\mathcal{A}$ running in time t and making at most q queries to its challenger $\mathcal{C}_f^{\mathsf{prf}}$, cannot distinguish f from a random function Rand with a non-negligible advantage.*

B The AP-AKA variant [18]

Protocol description. In 2003, Zhang proposed a variant of AKA he called AP-AKA, which we depict in Figure 5 (although we use a syntax closer to our own variant, to facilitate a comparison). Instead of the suite of seven cryptographic algorithms specified for the AKA protocol, Zhang only uses three independent functions F, G, H, which are all keyed with a key K. The authors do not specify what this key is in the AKA scenario, but considering the design of this protocol, it must be a function of the two keys $\mathsf{sk_C}$ and $\mathsf{sk_{Op}}$. We assume $K = \mathsf{sk_C}||\mathsf{sk_{Op}}$ in this case. For the security of the protocol, G must be a pseudorandom function (PRF), while F and H must be unforgeable MACs.



Instructions:

Client	Server	Operator
③: Recover $\mathsf{R}^{\{i\}}$, and $\mathsf{Autn}^{\{i\}}$. If $R_{\mathsf{C}}^{\{i\}}$ was used before by S: abort. Else: Find $1 \le i \le n$ s.t. $R_{\mathsf{C}}^{\{i\}} = H_K(i\|R_{\mathsf{C}})$ If no such i exists: abort; Else: check $\mathsf{Mac_S}$ for $i, \mathsf{R}^{\{i\}}, R_{\mathsf{C}}^{\{i\}}$ (or abort) Compute $\mathsf{Res} = F_K(\mathsf{R}^{\{i\}})$ Compute: $(\mathsf{CK}\|\mathsf{IK}) = G_K(\mathsf{R}^{\{i\}})$.	②: Store $\{\mathsf{AV}^{\{i\}}\}_{i=1}^{n}$. Choose next $\mathsf{AV}^{\{i\}}$, send related challenge ④: $\mathsf{Res} \stackrel{?}{=} \mathsf{Mac_C}$.	①: Check $\mathsf{Mac_2}$ w.r.t. $R_{\mathsf{fresh}}, R_{\mathsf{C}}$. If failure, send reject notice to S and abort; Else: for $i = 1, \ldots, n$, compute: $\mathsf{Mac_C} = F_K(\mathsf{R}^{\{i\}})$ $(\mathsf{CK}\|\mathsf{IK}) = G_K(\mathsf{R}^{\{i\}})$ $R_{\mathsf{C}}^{\{i\}} = H_K(i\|R_{\mathsf{C}})$ $\mathsf{Mac_S} = F_K(\mathsf{R}^{\{i\}}\|i\|R_{\mathsf{C}}^{\{i\}})$ $\mathsf{Autn}^{\{i\}} \leftarrow i\|R_{\mathsf{C}}^{\{i\}}\|\mathsf{Mac_S}$. $\mathsf{AV}^{\{i\}} := (\mathsf{R}^{\{i\}}, \mathsf{CK}, \mathsf{IK}, \mathsf{Autn}^{\{i\}}, \mathsf{Mac_C})$. End For.

Fig. 5. The AP-AKA Variant.

We first describe this protocol. The procedure begins by the same identification phase as the regular AKA procedure shown in Section 2.2. Namely, the

server sends an identification request, to which the client responds with either the permanent identifier IMSI or with a tuple consisting of the temporary identifier TMSI and the local area identifier LAI of the server that issued the TMSI.

The first modification made with respect to the classical AKA is extending the authentication vector request phase, which takes place between two parties in the original scheme, to three parties. The server/client communication takes place across an *insecure channel*, wheras the channel between the server and the operator is *secure*. Zhang [18] adds a message-exchange to the protocol every time the server needs fresh authentication vectors. This exchange is a typical challenge-response authentication: the server sends a fresh nonce R_{fresh}, and the client generates a fresh R_{C}, computing a MAC (namely the function F) keyed with the key K, on input the concatenation of $R_{\mathsf{fresh}}, R_{\mathsf{C}}$, and a unique server identifier $\mathsf{Id_S}$. The authentication vectors are similar to those in the original AKA, but they implicitly rely on the client's random value R_{C} and on fresh randomness $\mathsf{R}^{\{i\}}$ generated by the operator for $i = 1, \ldots, n$ (here n is the batch size). An initial step is to generate numbers $R_{\mathsf{C}}^{\{i\}}$ for $i = 1, \ldots, n$; these are generated by using the MAC function H on input i and R_{C}. The server authentication string $\mathsf{Mac_S}$ is computed as the function F on input the operator's randomness $\mathsf{R}^{\{i\}}$, the current index i, and a nonce $R_{\mathsf{C}}^{\{i\}}$ generated from R_{C}. The values $i, R_{\mathsf{C}}^{\{i\}}$, and $\mathsf{Mac_S}$ are grouped as $\mathsf{Autn}^{\{i\}}$ for each i. Each authentication vector $\mathsf{AV}^{\{i\}}$ consists of: the randomness $\mathsf{R}^{\{i\}}$, the authentication string $\mathsf{Autn}^{\{i\}}$, the session keys $(\mathsf{CK}, \mathsf{IK})$ which are derived as a result of the PRF G, keyed with K, on input the randomness $\mathsf{R}^{\{i\}}$, and the expected client-authentication string $\mathsf{Mac_C}$. Finally, a batch of n authentication vectors are sent to the server.

The remainder of the protocol proceeds analogously to the original AKA procedure, with the modifications imposed by the way the authentication vectors are generated. In particular, upon receiving the randomness $\mathsf{R}^{\{i\}}$ and the authentication string $\mathsf{Autn}^{\{i\}}$, the client verifies, in order: (1) that it has never received the same string $R_{\mathsf{C}}^{\{i\}}$; (2) that this value is consistent with the randomness R_{C}; (3) that the authentication $\mathsf{Mac_S}$ is correct with respect to this randomness. If any of these verifications fail, the client aborts. Else, it computes the session keys and its own authentication string Res, sending the latter to the server.

Stateful vs. Stateless. Zhang presented his variant as eliminating "dynamic states". We note that, while his work ensured that no sequence number is necessary, the protocol is not entirely stateless. In particular, the client must keep track of a list, which is dynamically updated, of already seen randomness $R_{\mathsf{C}}^{\{i\}}$ for a given nonce R_{C}. Although this makes the protocol stateful, it does eliminate the need to resynchronize the state of the two parties.

Security Problems. A first problem is the fact that the three functions F, G, and H use the same key. In particular, the values $\mathsf{Mac_S}$, $\mathsf{Mac_C}$, and $R_{\mathsf{C}}^{\{i\}}$, which are computed using the key K, are sent across an insecure channel. Since F and H are MACs, the confidentiality of the key K is not fully guaranteed; thus the guarantee of pseudorandomness of G is not sufficient to guarantee the

indistinguishability from random of the keys CK, IK. This weakness is remedied if all three functions are assumed to have pseudorandom output.

A more serious problem is a network-corruption attack, which is harder to prevent, and which originates in these two facts: (1) the new procedure to request authentication vectors originates from the server, not from the client (indeed, the client is not aware of whether the server still has pertinent authentication tokens or not); (2) the network-specific identifier $\mathsf{Id_S}$ is only used in the $\mathsf{Mac_2}$ value. In particular, the attack proceeds as follows:

1. The client C arrives in a vulnerable area (for which the server S^* will be corrupted). This server is authorized to request authentication tokens from the operator and does so. Then, S^* may use several such tokens with the client (but not all). We assume that there will be at least a single authentication vector AV which was not used. The adversary $\mathcal{A}$ then corrupts the server S^*, thereby also retrieving the vector AV.
2. The client leaves the vulnerable area to enter a non-vulnerable one (with an honest server S). The adversary $\mathcal{A}$ acts as a Man-in-the-Middle (MiM) between C and S. It blocks the message R_{fresh} and any other message sent by S, sending instead $\mathsf{R}, \mathsf{Autn}$ from the authentication vector AV.
3. The verifications on the client side pass as long as the client still retains its past value R_{C}. This is not specified exactly in the original paper [18], but considering that it is the *server* that initiates this exchange, it is likely that the client will not automatically replace R_{C} unless prompted by the server.

A possible countermeasure to this vulnerability is to ensure that once the client is aware of having moved from the area associated with S^*, it discards the old R_{C} and aborts unless it is asked to generate a new one.

Practical Aspects. The protocol presented by Zhang [18] is not fully specified, but it does not follow closely the practical constraints of mobile networks. The protocol forces a lot of complexity on the client, which has to verify the uniqueness of the nonce $R_{\mathsf{C}}^{\{i\}}$, to search exhaustively for the correct index which gives an (honestly generated) $R_{\mathsf{C}}^{\{i\}}$, and to generate the randomness R_{C}. Since $R_{\mathsf{C}}^{\{i\}}$ is generated given the secret key K, it must be computed securely: it is not possible to delegate this computation to the phone (which is a more powerful, but untrusted tool). We reiterate that the rationale of the initial AKA design was that the client's SIM card could not generate its own randomness.

Another concern is the lack of specificity with respect to the client and operator keys, which are replaced by the generic key K. The fact that this key is used in MAC functions exposes the keys to attackers, as explained in the first attack above. In our results, we prove that for both TUAK and $\mathsf{MILENAGE}$ the seven cryptographic algorithms are PRFs with respect to both the client and the operator keys; this is a much stronger result. The same lack of specificity affects the keys CK, IK, which are generically denoted as a secret key SK in the paper of Zhang. We note that the latter is a more sounder cryptographic design, since in the AKA protocol, the two keys are output by *different* PRFs, but on

the same input. In particular, a stronger property is required than merely the pseudorandomness of the two concerned algorithms: the two values must be independent even for adversarially-controlled input. We show that this is the case for MILENAGE and TUAK, nonetheless.

C Full protocol description

In the AKA protocol [3,4], mutual client-backend authentication is provided using Message Authentication Codes (MAC) computed by three of the TUAK algorithms, while the secret keys are derived from a random value and a shared secret key with a key derivation function (KDF), by means of the rest of the TUAK functions.

The basic framework is a challenge-response stateful protocol between two main actors: the HLR (Home Location Register) and the ME/USIM (Mobile Equipment/User Subscriber Identity Module). This protocol needs an intermediate entity, the VLR (Visited Location Register), as specified in Section **??**. Both the ME/USIM and the HLR keep track of counters, denoted respectively $\mathsf{Sqn_C}$ and $\mathsf{Sqn_{HLR}}$; these sequence numbers are meant to provide entropy and enable network authentication (from HLR to ME/USIM). Technically, one can view the user's sequence number as an increasing counter, while the latter keeps track of the highest authenticated counter the user has accepted.

The AKA protocol uses a set of seven functions: $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$, $\mathcal{F}_4$, $\mathcal{F}_5$, $\mathcal{F}_1^*$, $\mathcal{F}_5^*$. The first two are used to authenticate a MAC answer, proving that both participants know the same subscriber key $\mathsf{sk_C}$ and the same operator key $\mathsf{sk_{Op}}$. Algorithm $\mathcal{F}_1$ is called the network authentication function. As its name implies, it allows the subscriber to authenticate the network. Furthermore, this function provides the data integrity used to derive keys (in particular authenticating the random, session-specific value R). Algorithm $\mathcal{F}_2$ is called the subscriber authentication function, and it allows the network to authenticate the subscriber C by proving that the entity owns the subscriber key $\mathsf{sk_C}$ and the operator key $\mathsf{sk_{Op}}$.

The following three algorithms, $\mathcal{F}_3, \ldots, \mathcal{F}_5$, are used as key derivation functions, outputting respectively a cipher key (CK), an integrity key (IK), and an anonymity key (AK), all derived on input the subscriber key $\mathsf{sk_C}$, the operator key $\mathsf{sk_{Op}}$, and the session-specific random value R. Notice that the master key $\mathsf{sk_C}$ is only known by HLR and ME/USIM, but not by the intermediate entity VLR.

The last key, AK, is used to mask the sequence number Sqn, but it is not part of the session keys. Its function is to blind the value of Sqn since the latter may leak some information about the subscriber. In order to ensure that no long-term desynchronization occurs, the AKA protocol provides a re-synchronization procedure between the two participants, in which the user forces a new sequence number on the backend server, using the $\mathcal{F}_1^*$ and $\mathcal{F}_5^*$ to authenticate this value much in the same way that the terminal has authenticated its own sequence number and random value. Figure 6 details the challenge-response of AKA procedure.

The operator key. Subscribers to the same operator all share the operator's own secret key, in practice a 256-bit integer. This value is not directly stored on the phone, but rather an intermediate value, obtained by running the internal Keccak permutation on input $\mathsf{sk_{Op}}$ and several constants, is embedded in the SIM card. Thus, whereas this value enters in all future runs of the cryptographic algorithms, it is never stored in clear on the user's mobile.

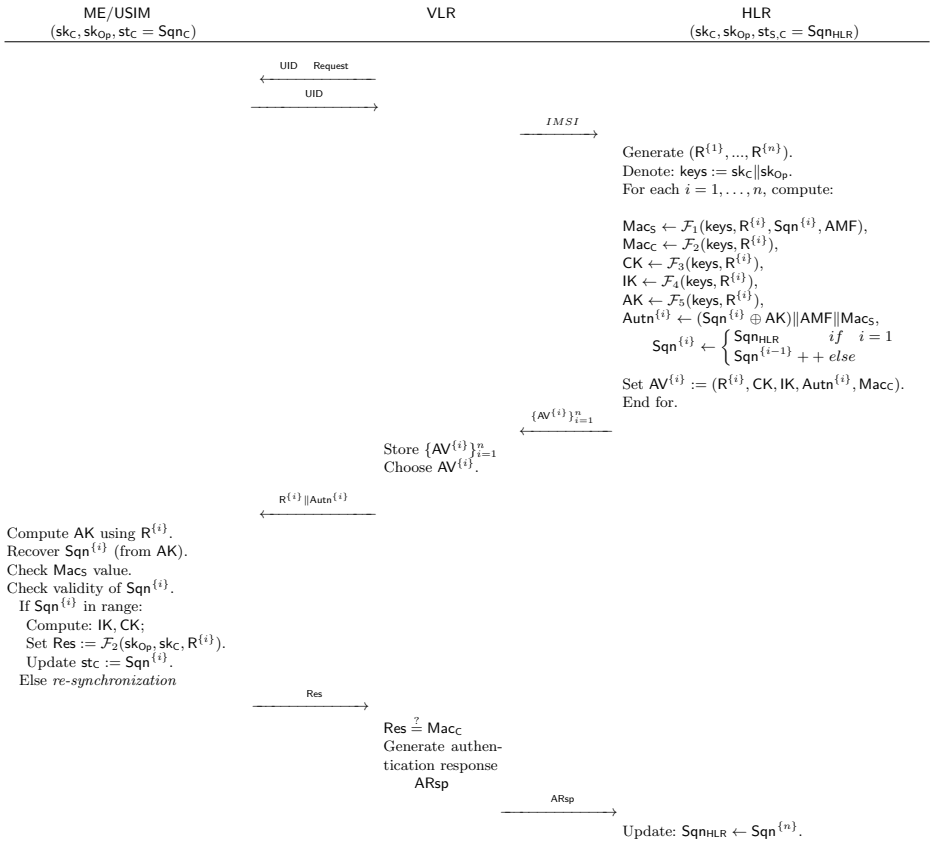


Fig. 6. The AKA Procedure.

IMSI, TMSI, UID. Globally, the procedure starts when the user equipment switches on. To identify the ME/USIM to the VLR, the mobile equipment receives a user equipment request and responds to the VLR, in clear text, with a UID. This value can be either an IMSI (International Mobile Subscriber Identify) or a TMSI (Temporary Mobile Subscriber Identity) which is a value exchanged between

the VLR and the subscriber during a previous session where both entities are mutually authenticated.

These TMSI are exchanged in order to guarantee the uniqueness of the user equipment request during following sessions. In practice, the IMSI is used either for the first session or when the serving network cannot retrieve the IMSI from the temporary identity. Then, the VLR forwards the IMSI of the subscriber to the HLR.

Challenge-Response. After receiving the IMSI, the HLR generates a fresh sequence number Sqn and an unpredictable variable R. By using the subscriber's key $\mathsf{sk_C}$ and the corresponding operator key $\mathsf{sk_{Op}}$, it then generates a list of n unique authentication vectors AV composed of five strings: R, $\mathsf{Mac_C}$, CK, IK, Autn. For every authentication vector, the sequence number is updated. The update procedure depends on the chosen method. The specifications feature a first method which does not take into account the notion of time, and which basically increments by 1 the most significant 32-first value of the sequence number. A second and third subsequent methods feature a time-based sequence number update based on a clock giving universal time [4]. The authentication vector is generated as follows:

$$\begin{aligned}
\mathsf{Mac_S} &\leftarrow \mathcal{F}_1(\mathsf{sk_C}, \mathsf{sk_{Op}}, \mathsf{R}, \mathsf{Sqn}, \mathsf{AMF}),\\
\mathsf{Mac_C} &\leftarrow \mathcal{F}_2(\mathsf{sk_C}, \mathsf{sk_{Op}}, \mathsf{R}),\\
\mathsf{CK} &\leftarrow \mathcal{F}_3(\mathsf{sk_C}, \mathsf{sk_{Op}}, \mathsf{R}),\\
\mathsf{IK} &\leftarrow \mathcal{F}_4(\mathsf{sk_C}, \mathsf{sk_{Op}}, \mathsf{R}),\\
\mathsf{AK} &\leftarrow \mathcal{F}_5(\mathsf{sk_C}, \mathsf{sk_{Op}}, \mathsf{R}),\\
\mathsf{Autn} &\leftarrow (\mathsf{Sqn} \oplus \mathsf{AK}) \| \mathsf{AMF} \| \mathsf{Mac_S},
\end{aligned}$$

where $\mathsf{Mac_S}$ is the message authentication code of the network by the subscriber, $\mathsf{Mac_C}$ is the message authentication code of the subscriber by the network and AMF the authentication and key management field (which is a known, public constant).

The HLR sends the list of the authentication vectors AV to the VLR. This list may also contain only a single authentication vector. Upon the reception and storage of these vectors, when the VLR initiates an authentication and key agreement, it selects the next authentication vector from the ordered array and stores $\mathsf{Mac_C}$ and the session keys CK and IK. Then, it forwards (R, Autn) to ME/USIM.

The ME/USIM verifies the freshness of the received authentication token. To this end, it recovers the sequence number by computing the anonymity key AK which in its own turn depends on three values: $\mathsf{sk_C}$, $\mathsf{sk_{Op}}$, and the received R. Then, the user verifies the received $\mathsf{Mac_S}$ computing $\mathcal{F}_1(\mathsf{sk_C}$, $\mathsf{sk_{Op}}$, R, Sqn , AMF) with the received value R and the Sqn. If they are different, the user sends *authentication failure* message back to the VLR and the user abandons the procedure. In case the execution is not aborted, the ME/USIM verifies if the

received Sqn value is in a correct range relatively to a stored value $\mathsf{Sqn_C}$[7]. If the Sqn is out of range, the user sends a *synchronization failure* message back to the VLR, which triggers a *re-synchronization procedure*, depicted further in Figure 7.

The $\mathsf{Mac_S}$ value does not only ensure integrity, but also the authentication of the network by ME/USIM. If the two previous verifications are successful i.e if the received authentication token is fresh, the network is authenticated by the ME/USIM. Then, the ME/USIM computes CK, IK and $\mathsf{Res} \leftarrow \mathcal{F}_2(\mathsf{sk_C}, \mathsf{sk_{Op}}, \mathsf{R})$. To improve efficiency, Res, CK, and IK could also be computed earlier, at the same time that AK is computed. Finally, the user sends Res to VLR. If $\mathsf{Res} = \mathsf{Mac_C}$, the VLR successfully authenticates the ME/USIM. Otherwise, the VLR will initiate an *authentication failure* report procedure with the HLR. Note that the verification of the sequence number by the ME/USIM will cause the rejection of any attempt to re-use an authentication token more than once.

Re-synchronizing. The re-synchronization procedure is used when the subscriber detects that the received sequence number is not in the correct range, but that it has been correctly authenticated. The single goal of this procedure is the re-initialization of the sequence number, and does not imply immediately any mutual authentication or key agreement (rather it triggers a new authentication attempt).

Indeed, the ME/USIM sends an synchronization failure message, consisting of a parameter Auts, with

$$\mathsf{Auts} = (\mathsf{Sqn_C} \oplus \mathsf{AK}^*) \| \mathsf{Mac}^*$$

where the key is computed as $\mathsf{Mac}^* = \mathcal{F}_1^*(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Sqn_C}, \mathsf{AMF})$ and $\mathsf{AK}^* = \mathcal{F}_5^*(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R})$.

The $\mathcal{F}_1^*$ algorithm is a MAC function with the additional property that no valuable information can be inferred from Mac^* (in particular this function acts as a PRF). Though similar to $\mathcal{F}_1$, the $\mathcal{F}_1^*$ algorithm is designed so that the value Auts cannot be replayed relying on the output of $\mathcal{F}_1$. Furthermore, the anonymity key generated by the client in the resynchronization is obtained via the $\mathcal{F}_5^*$ algorithm rather than by $\mathcal{F}_5$, even if the same random value R is used.

Upon receiving a re-synchronization failure message, the VLR does not immediately send a new user authentication request to the ME/USIM, but rather notifies the HLR of the re-synchronization failure, sending the parameter Auts and the session-specific R. When the HLR receives this answer, it creates a new batch of authentication vectors. Depending on whether the retrieved, authenticated Sqn indicates that the HLR's sequence number is out of range or not, the backend server either starts from the last authenticated sequence number, or updates the latter to the user's sequence number.

More precisely, the HLR retrieves the $\mathsf{Sqn_C}$ by computing $\mathcal{F}_5^*(\mathsf{sk_C}, \mathsf{sk_{Op}}, \mathsf{R}) \oplus \lfloor \mathsf{Auts} \rfloor_{48}$. Then, it verifies if the incremented $\mathsf{Sqn_{HLR}}$ is in the correct range relatively to $\mathsf{Sqn_C}$. If the $\mathsf{Sqn_{HLR}}$ verifies this property, it sends a new list of authentication data vectors initiated with $\mathsf{Sqn_{HLR}}$ else HLR verifies the value of

[7] The sequence number Sqn is considered to be in the correct range relatively to $\mathsf{Sqn_C}$ if and only if $\mathsf{Sqn} \in (\mathsf{Sqn_C}, \mathsf{Sqn_C} + \Delta)$, where Δ is defined by the operator.

Mac^*İf this step is successful, it resets the value of $\mathsf{Sqn_{HLR}}$ to $\mathsf{Sqn_{HLR}} := \mathsf{Sqn_C}$ and sends a new list of authentication data vectors initiated with this updated $\mathsf{Sqn_{HLR}}$. This list may also contain only a single authentication vector. Figure 7 details this re-synchronization procedure.

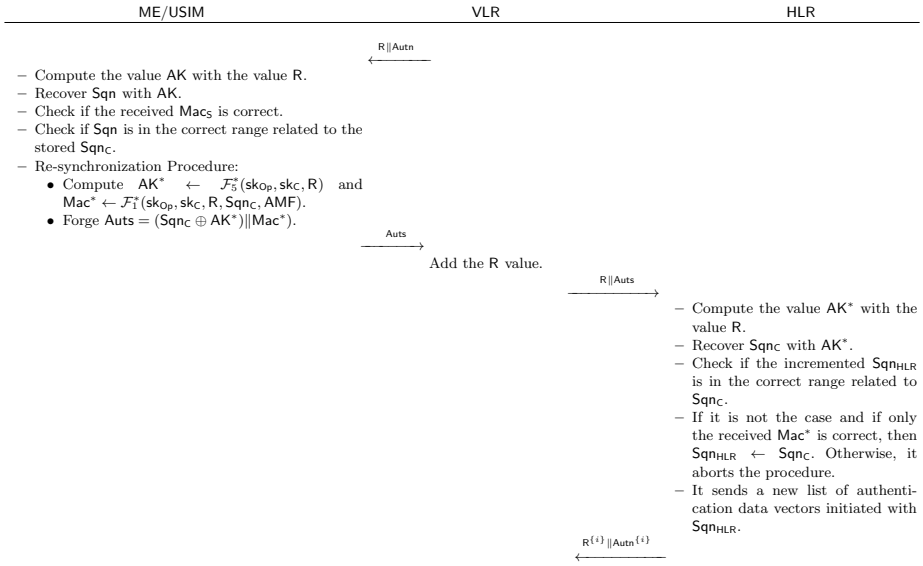


Fig. 7. The re-synchronization procedure of AKA protocol.

C.1 The TUAK algorithms

TUAK [2] is a set of algorithms based on a truncation of the internal permutation function of Keccak; however, for efficiency reasons, only one or two iterations of the internal TUAK permutation are used. The goal of the TUAK functions is to provide secure authentication and key-exchange in the AKA protocol. In particular the TUAK functions $\mathcal{F}_1$ (respectively $\mathcal{F}_1^*$) and $\mathcal{F}_2$ must provide authentication, while $\mathcal{F}_3$, $\mathcal{F}_4$, and $\mathcal{F}_5$ (respectively $\mathcal{F}_5^*$) are used to derive the session keys used to attain confidentiality, integrity, and anonymity.

The seven functions are parametrized by:

- Inputs: $\mathsf{sk_{Op}}$ a 256-bit long term operator key, a 128-bit random value R, a 48-bit sequence number Sqn, and a 16-bit authentication field management string AMF chosen by the operator (the last two values are only used for the MAC generation). Note that all subscribers to the same operator will share that operator's key $\mathsf{sk_{Op}}$.
- A subscriber key $\mathsf{sk_C}$ shared out of band between the HLR and ME/USIM allows to initialize the value Key:

- If $|\mathsf{sk_C}| = 128$ bits, then $\mathsf{Key} \leftarrow \mathsf{sk_C}[127..0] \| 0^{128}$.
- If $|\mathsf{sk_C}| = 256$ bits, then $\mathsf{Key} \leftarrow \mathsf{sk_C}[255..0]$.

– Several public constants:
- AN: a fixed 56-bit value $0x5455414B312E30$.
- Inst and Inst' are fixed binary variables of 8 bits, specified in [2], which depend on the functions and the output sizes.

The generation of MAC's or derived key starts similarly by initializing a value Top_C. To do so, one applies a first $\mathsf{f_{Keccak}}$ permutation on a 1600-bit state Val_1 as follows:

$$\mathsf{Val}_1 = \mathsf{sk_{Op}} \| \mathsf{Inst} \| \mathsf{AN} \| 0^{192} \| \mathsf{Key} \| \mathsf{Pad} \| 1 \| 0^{512},$$

where Pad is a bitstring output by a padding function. The value Top_C corresponds to the first 256 bits of this output.

At this point, the behavior of the functions $\mathcal{F}_1$ and $\mathcal{F}_1^*$ diverges from that of the other functions. To generate the MAC value of $\mathcal{F}_1$ and $\mathcal{F}_1^*$, we take as input Sqn, AMF and R, three values chosen by the operator, and some constants. After the generation of Top_C, we initialize a second state, namely,

$$\mathsf{Val}_2 = \mathsf{Top}_C \| \mathsf{Inst}' \| \mathsf{AN} \| \mathsf{R} \| \mathsf{AMF} \| \mathsf{Sqn} \| \mathsf{Key} \| \mathsf{Pad} \| 1 \| 0^{512}.$$

Then, one applies the TUAK permutation on Val_2, using only the first 64 bits to compute $\mathsf{Mac_S}$. To generate the session keys and run $\mathcal{F}_2$, one initializes a second state for this function, namely,

$$\mathsf{Val}_2 = \mathsf{Top}_C \| \mathsf{Inst}' \| \mathsf{AN} \| \mathsf{R} \| 0^{63} \| \mathsf{Key} \| \mathsf{Pad} \| 1 \| 0^{512}.$$

Then, the TUAK permutation is applied on Val_2 yielding Out, which in turn is used to compute the response $\mathsf{Mac_C}$ and the session keys:

$$\begin{aligned} \mathsf{Mac_C} &= \lfloor \mathsf{Out} \rfloor_{|\ell|-1..0}, \ell \in \{16, 32, 64, 128\}, \\ \mathsf{CK} &= \lfloor \mathsf{Out} \rfloor_{256..384} \text{ and } |\mathsf{CK}| = 128, \\ \mathsf{IK} &= \lfloor \mathsf{Out} \rfloor_{512..640} \text{ and } |\mathsf{IK}| = 128, \\ \mathsf{AK} &= \lfloor \mathsf{Out} \rfloor_{768..816} \text{ and } |\mathsf{AK}| = 48. \end{aligned}$$

This is also depicted in Figure 8.

The way the output of the functions is truncated and used is the reason why TUAK is called a *multi-output function*. This is one of TUAK's chief differences from MILENAGE and has a no-negligible impact on its efficiency, as it saves a few calls of the internal function. However, this multi-output property can be an issue for the security of the master key, since during one session we can have as many as four calls to the same function with similar inputs (and a different truncation). Having different chunks of the same 1600-bit state (called Out in

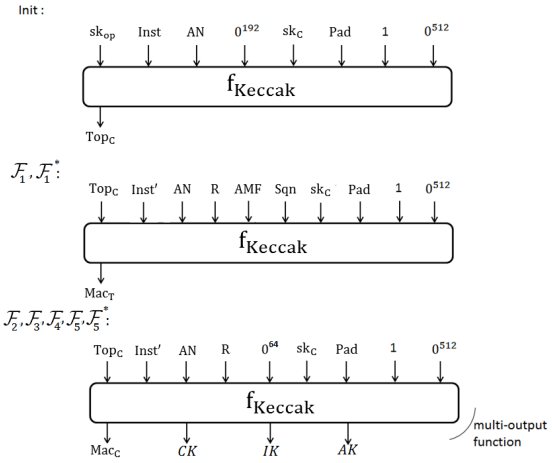


Fig. 8. TUAK diagram.

our description) can lead to recovering the long-term key $\mathsf{sk_C}$ by the reversibility of the TUAK permutation. The concatenation of all the different chunks used per session totals at most only 432 out of the 1600 output bits. Thus, though having multiple outputs can be hazardous in general, the Keccak-based construction of TUAK allows this without compromising the long-term parameters.

C.2 MILENAGE Algorithms

Milenage algorithms: MILENAGE [1] is a set of algorithms which aims to achieve authentication and key generation properties. As opposed to TUAK which is based on Keccak's internal permutation, the MILENAGE algorithms are based on the Advanced Standard Protocol (AES).

The functions $\mathcal{F}_1^*$ and $\mathcal{F}_2^*$ must provide authentication while the functions $\mathcal{F}_3^*$, $\mathcal{F}_4^*$ and $\mathcal{F}_5^*$ are used to derive key material in order to achieve confidentiality, integrity and anonymity. The different parameters of these functions are:

- Inputs: $\mathsf{sk_{Op}}$ a 128-bit long term credential key that is fixed by the operator, a 128-bit random value R , a 48-bit sequence number Sqn and a 16-bit authentication field management AMF chosen by the operator (the last two values are only used for the MAC generation). We denote that the subscriber key $\mathsf{sk_{Op}}$ is a private key shared by all the subscriber of the same operator. Consequently, we do not consider $\mathsf{sk_{Op}}$ as a private key.
- A 128-bit subscriber key $\mathsf{sk_C}$ shared out of band between the HLR and ME/USIM.
- Five 128-bit constants c_1,c_2,c_3,c_4,c_5 which are Xored onto intermediate variables and are defined as follows:
  - $\mathsf{c}_1[i] = 0, \forall i \in \{0, 127\}$.

- $\mathsf{c}_2[i] = 0, \forall i \in \{0, 127\}$, except that $\mathsf{c}_2[127] = 1$.
- $\mathsf{c}_3[i] = 0, \forall i \in \{0, 127\}$, except that $\mathsf{c}_3[126] = 1$.
- $\mathsf{c}_4[i] = 0, \forall i \in \{0, 127\}$, except that $\mathsf{c}_4[125] = 1$.
- $\mathsf{c}_5[i] = 0, \forall i \in \{0, 127\}$, except that $\mathsf{c}_5[124] = 1$.

– Five integers r_1,r_2,r_3,r_4,r_5 in the range $\{0, 127\}$ which define amounts by which intermediate variables are cyclically rotated and are defined as follows: $\mathsf{r}_1 = 64; \mathsf{r}_2 = 0; \mathsf{r}_3 = 32; \mathsf{r}_4 = 64; \mathsf{r}_5 = 96$.

The generation of MAC's or derived key starts similarly by initializing a value Top_C. To do so, one applies a first called of the well-known function AES on inputs the operator and subscriber keys such as:

$$\mathsf{Top}_C = \mathsf{sk}_{\mathsf{Op}} \oplus \mathsf{AES}_{\mathsf{sk_c}}(\mathsf{sk}_{\mathsf{Op}})$$

. We recall that, $\mathsf{AES}_K(M)$ denotes the result of applying the Advanced Encryption Standard encryption algorithm to the 128-bit value M under the 128-bit key K. Then, we compute the following values taking as input Sqn, R, AMF and others constants:

– $\mathsf{Temp} = \mathsf{AES}_{\mathsf{sk_c}}(\mathsf{R} \oplus \mathsf{Top}_C)$,
– $\mathsf{Out}_1 = \mathsf{AES}_{\mathsf{sk_c}}(\mathsf{Temp} \oplus \mathsf{Rot}_{\mathsf{r}_1}(\mathsf{Sqn} \| \mathsf{AMF} \| \mathsf{Sqn} \| \mathsf{AMF}) \oplus \mathsf{c}_1) \oplus \mathsf{Top}_C$,
– $\mathsf{Out}_2 = \mathsf{AES}_{\mathsf{sk_c}}(\mathsf{Rot}_{\mathsf{r}_2}(\mathsf{Temp} \oplus \mathsf{Top}_C) \oplus \mathsf{c}_2) \oplus \mathsf{Top}_C$,
– $\mathsf{Out}_3 = \mathsf{AES}_{\mathsf{sk_c}}(\mathsf{Rot}_{\mathsf{r}_3}(\mathsf{Temp} \oplus \mathsf{Top}_C) \oplus \mathsf{c}_3) \oplus \mathsf{Top}_C$,
– $\mathsf{Out}_4 = \mathsf{AES}_{\mathsf{sk_c}}(\mathsf{Rot}_{\mathsf{r}_4}(\mathsf{Temp} \oplus \mathsf{Top}_C) \oplus \mathsf{c}_4) \oplus \mathsf{Top}_C$,
– $\mathsf{Out}_5 = \mathsf{AES}_{\mathsf{sk_c}}(\mathsf{Rot}_{\mathsf{r}_5}(\mathsf{Temp} \oplus \mathsf{Top}_C, \mathsf{r}_5) \oplus \mathsf{c}_5) \oplus \mathsf{Top}_C$.

All the outputs of the $\mathsf{MILENAGE}$ algorithms are computed as follows:

– **Output** $\mathcal{F}_1$**:** $\mathsf{Mac_C} = \lfloor \mathsf{Out}_1 \rfloor_{0..63}$,
– **Output** $\mathcal{F}_1^*$**:** $\mathsf{Mac}^* = \lfloor \mathsf{Out}_1 \rfloor_{64..127}$,
– **Output** $\mathcal{F}_2$**:** $\mathsf{Mac_S} = \lfloor \mathsf{Out}_2 \rfloor_{64..127}$,
– **Output** $\mathcal{F}_3$**:** $\mathsf{CK} = \mathsf{Out}_3$,
– **Output** $\mathcal{F}_4$**:** $\mathsf{IK} = \mathsf{Out}_4$,
– **Output** $\mathcal{F}_5$**:** $\mathsf{AK} = \lfloor \mathsf{Out}_2 \rfloor_{0..47}$,
– **Output** $\mathcal{F}_5^*$**:** $\mathsf{AK}^* = \lfloor \mathsf{Out}_5 \rfloor_{0..47}$,

This is also described in figure 9

D Full Proofs

Theorem 8. *[*$\mathsf{W.K.Ind}$*-resistance.] Let* $G : \{0,1\}^\kappa \times \{0,1\}^d \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^n$ *be our specified function specified in section 4 and* Π *the* AKA *protocol specified in section 4. Consider a* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_G)$*-adversary* $\mathcal{A}$ *against the* $\mathsf{W.K.Ind}$*-security of the protocol* Π*, running in time* t *and creating at most* q_{exec} *party instances with at most* q_{res} *resynchronizations per instance, and making* q_G *queries to the function* G*. Denote the advantage of this adversary as* $\mathsf{Adv}_\Pi^{\mathsf{W.K.Ind}}(\mathcal{A})$*. Then there exists a* $(t' \approx O(t), q' = q_G + q_{\mathsf{exec}}(2 + q_{\mathsf{res}}))$*-prf-adversary* $\mathcal{A}'$ *on* G *such that:*

$$\mathsf{Adv}_\Pi^{\mathsf{W.K.Ind}}(\mathcal{A}) \leq n_{\mathsf{C}} \cdot \left(\frac{q_{\mathsf{exec}}^2}{2^{|\mathsf{R}|}} + \mathsf{Adv}_G^{\mathsf{prf}}(\mathcal{A}') \right).$$

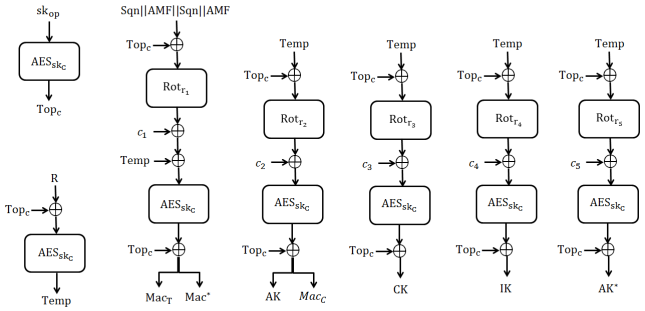


Fig. 9. MILENAGE diagram.

Proof. Our proof has the following hops.

Game $\mathbb{G}_0$: This game works as the $\mathsf{W.K.Ind}$-game stipulated in our security model 3. The goal of the adversary $\mathcal{A}_{\mathbb{G}_0}$ is to distinguish, for a fresh instance that ends in an accepting state, the fresh session keys from random ones.

Game $\mathbb{G}_1$: We modify $\mathbb{G}_0$ to only consider the new query $\mathsf{Corrupt}(\mathsf{P}, \mathtt{type})$ but keeping the same goal. We note that this new query permits to consider the corruption of the key operator independently to the corruption of the subscriber keys. This new query behaves as follows:

$\mathsf{Corrupt}(\mathsf{P}, \mathtt{type})$: yields to the adversary the long-term keys of party $\mathsf{P} \neq \mathsf{S}$ (else, if the oracle takes as input $\mathsf{P} = \mathsf{S}$, then it behaves as usual calling the oracle **OpAccess**). The output of the oracle depends on the value $\mathtt{type} \in \{\mathtt{sub}, \mathtt{op}, \mathsf{all}\}$. If $\mathtt{type} = \mathtt{sub}$, then the returned value is sk_P. If $\mathtt{type} = \mathtt{op}$, then the oracle returns $\mathsf{sk}_{\mathsf{Op}}$. Then, for $\mathtt{type} = \mathsf{all}$, we return the both values sk_P, $\mathsf{sk}_{\mathsf{Op}}$. If $\mathtt{type} \in \{\mathtt{sub}, \mathsf{all}\}$, then P (and all its instances, past, present, or future), are considered to be adversarially controlled.

We argue that given any adversary $\mathcal{A}$ playing the game $\mathbb{G}_1$ and winning w.p. $\epsilon_\mathcal{A}$, the same adversary wins the game $\mathbb{G}_0$ w.p. at least $\epsilon_\mathcal{A}$ (this is trivial since in game $\mathbb{G}_1$, $\mathcal{A}$ has more information).

$$\Pr[\mathcal{A} \text{ wins } \mathbb{G}_0] \leq \Pr[\mathcal{A} \text{ wins } \mathbb{G}_1].$$

Game $\mathbb{G}_2$: We modify $\mathbb{G}_1$ to only allow interactions with a single client (any future $\mathsf{CreateCl}$ calls for a client would be answered with an error symbol $\perp$). The challenger generates only a single operator key, which is associated with the operator chosen for the registered client and chooses a bit $b \in \{0, 1\}$. We process

as follows: for any adversary $\mathcal{A}_{\mathbb{G}_1}$ winning the game $\mathbb{G}_1$ with a no-negligible success probability $\epsilon\mathcal{A}_{\mathbb{G}_1}$. we propose to construct an adversary $\mathcal{A}_{\mathbb{G}_2}$ winning the game $\mathbb{G}_2$ with a black-box access to the adversary $\mathcal{A}_{\mathbb{G}_1}$.

Adversary $\mathcal{A}_{\mathbb{G}_2}$ begins by choosing a single client C. For every user registration request that $\mathcal{A}_{\mathbb{G}_1}$ sends to its challenger, $\mathcal{A}_{\mathbb{G}_2}$ responds as follows: if the registered client is C, then it forwards the exact $\mathsf{CreateCl}$ query that $\mathcal{A}_{\mathbb{G}_1}$ makes to its own $\mathsf{CreateCl}$ oracle. Else, if $\mathcal{A}_{\mathbb{G}_1}$ registers any client $\mathsf{C}^* \neq \mathsf{C}$, $\mathcal{A}_{\mathbb{G}_2}$ simulates the registration, generating $\mathsf{sk}_{\mathsf{C}^*}$ and $\mathsf{Sqn}_{\mathsf{C}^*}$, returning the latter value. Adversary $\mathcal{A}_{\mathbb{G}_2}$ also generates $n_{\mathsf{Op}} - 1$ operator keys, and associates them with the clients as follows: the target client C is associated with the same operator given as input by $\mathcal{A}_{\mathbb{G}_1}$ to the $\mathsf{CreateCl}$ query (thus with the operator key $\mathsf{sk}_{\mathsf{Op}}$ generated by the challenger of game $\mathbb{G}_2$). Let this target operator be denoted as Op. Adversary $\mathcal{A}_{\mathbb{G}_2}$ queries $\mathsf{Corrupt}(\mathsf{C}, \mathsf{op})$ and stores $\mathsf{sk}_{\mathsf{Op}}$.

We distinguish between two types of other clients. For all other clients C^* which are registered by $\mathcal{A}_{\mathbb{G}_1}$ with an operator $\mathsf{Op}^* \neq \mathsf{Op}$, adversary $\mathcal{A}_{\mathbb{G}_2}$ associates Op^* with one of its generated keys $\mathsf{rsk}_{\mathsf{Op}^*}$. Recall that, since adversary $\mathcal{A}_{\mathbb{G}_1}$ plays the game in the presence of n_{Op} operators, there are $n_{\mathsf{Op}} - 1$ keys which will be used this way. We call all clients $\mathsf{C}^* \neq \mathsf{C}$ registered by $\mathcal{A}_{\mathbb{G}_0}$ with the target operator Op the *brothers* of the target client C. Adversary $\mathcal{A}_{\mathbb{G}_2}$ associates each brother of C with the corrupted key $\mathsf{sk}_{\mathsf{Op}}$ it learns from its challenger.

In the rest of the simulation, whenever $\mathcal{A}_{\mathbb{G}_1}$ makes a query to an instance of some party C^*, not a brother of C, the adversary $\mathcal{A}_{\mathbb{G}_2}$ simulates the response using the values $\mathsf{sk}_{\mathsf{C}^*}$, $\mathsf{rsk}_{\mathsf{Op}^*}$, and the current value of Sqn. For the brothers of C, the simulation is done with $\mathsf{sk}_{\mathsf{C}^*}$, $\mathsf{sk}_{\mathsf{Op}}$, and the current Sqn. For the target client C, any queries are forwarded by $\mathcal{A}_{\mathbb{G}_2}$ to its challenger.

Any corruption or reveal queries are dealt with in a similar way. Note that $\mathcal{A}_{\mathbb{G}_2}$ cannot query $\mathsf{Corrupt}$ to its adversary (this is a condition of freshness). The simulation is thus perfect up to the Test query.

In the Test query, $\mathcal{A}_{\mathbb{G}_1}$ chooses a $\mathsf{fresh\ session}$ and sends it to $\mathcal{A}_{\mathbb{G}_2}$ (acting as a challenger). Note that $\mathcal{A}_{\mathbb{G}_2}$ will be able to test whether this instance is fresh, as freshness is defined in terms of $\mathcal{A}_{\mathbb{G}_1}$'s queries. If $\mathcal{A}_{\mathbb{G}_1}$ queries Test with a client other than the target client C, then $\mathcal{A}_{\mathbb{G}_2}$ aborts the simulation, tests a random, fresh instance of the client C (creating one if necessary), and guesses the bit d, winning with probability at least $\frac{1}{2}$. Else, if $\mathcal{A}_{\mathbb{G}_1}$ queried a fresh instance of C, $\mathcal{A}_{\mathbb{G}_2}$ forwards this choice to its challenger and receives the challenger's input. The adversary $\mathcal{A}_{\mathbb{G}_2}$ forwards the input of the challenger to $\mathcal{A}_{\mathbb{G}_1}$ and then receives $\mathcal{A}$'s output d, which will be $\mathcal{A}_{\mathbb{G}_2}$'s own response to its own challenger.

Denote by E_1 the event that adversary tests C in game $\mathbb{G}_1$, while $\bar{\mathsf{E}_1}$ denotes the event that $\mathcal{A}_{\mathbb{G}_1}$ chooses to test $\mathsf{C}^* \neq \mathsf{C}$.

It holds that:

$$\begin{aligned}
\Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}] &= \Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins} \mid \mathsf{E}_1] \cdot \Pr[\mathsf{E}_1] + \Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins} \mid \bar{\mathsf{E}_1}] \cdot \Pr[\bar{\mathsf{E}_1}] \\
&\geq \frac{1}{n_{\mathsf{C}}} \Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins}] + \frac{1}{2} \cdot \big(1 - \frac{1}{n_{\mathsf{C}}}\big) \\
&\geq \frac{1}{n_{\mathsf{C}}} \Pr[\mathcal{A}_{\mathbb{G}_0}\mathsf{wins}] + \frac{1}{2} \cdot \big(1 - \frac{1}{n_{\mathsf{C}}}\big).
\end{aligned}$$

Note that adversary $\mathcal{A}_{\mathbb{G}_2}$ makes one extra query with respect to $\mathcal{A}_{\mathbb{G}_1}$, since we need to learn the key of the target operator.

Game $\mathbb{G}_3$: We modify $\mathbb{G}_2$ to ensure that the random values sampled by honest server instances are always unique.

This gives us a security loss (related to the respective collisions between the R in two different instances) of

$$\left| \Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}] \right| \leq \frac{q_{\mathsf{exec}}^2}{2^{|\mathsf{R}|}}.$$

Game $\mathbb{G}_4$: We modify $\mathbb{G}_3$ to replace outputs of the internal cryptographic functions by truly random, but consistent values (they are independent of the input, but the same input gives the same output). We argue that the security loss is precisely the advantage of the adversary $\mathcal{A}$ against the pseudorandomness of function G. Note that the total number of queries to the related functions are at most 2 G per honest instance(thus totaling at most $\mathsf{q}_G + q_{\mathsf{exec}}(2 + q_{\mathsf{res}})$ queries to the function G).

$$\left| \Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_4}\mathsf{wins}] \right| \leq \mathsf{Adv}_G^{\mathsf{prf}}(\mathcal{A}).$$

Winning $\mathbb{G}_4$: At this point, the adversary plays a game in the presence of a single client C. The goal of this adversary is to distinguish a random session key to a fresh session key. But, in game $\mathbb{G}_4$, queries to G return truly random, consistent values. In this case, the adversary can do no better than guessing. Thus, we have:

$$\Pr[\mathcal{A}_{\mathbb{G}_4}\mathsf{wins}] = \frac{1}{2}.$$

Security statement: This yields the following result:

$$\begin{aligned}
\frac{1}{n_{\mathsf{C}}} \cdot \Pr[\mathcal{A}_{\mathbb{G}_0}\mathsf{wins}] + \frac{1}{2} \cdot (1 - \frac{1}{n_{\mathsf{C}}}) &\leq \frac{q_{\mathsf{exec}}^2}{2^{|\mathsf{R}|}} + \mathsf{Adv}_G^{\mathsf{prf}}(\mathcal{A}) \\
\Leftrightarrow \frac{1}{n_{\mathsf{C}}} \cdot \mathsf{Adv}_{\Pi}^{\mathsf{W.K.Ind}}(\mathcal{A}_{\mathbb{G}_0}) &\leq \frac{q_{\mathsf{exec}}^2}{2^{|\mathsf{R}|}} + \mathsf{Adv}_G^{\mathsf{prf}}(\mathcal{A}) \\
\Leftrightarrow \mathsf{Adv}_{\Pi}^{\mathsf{W.K.Ind}}(\mathcal{A}_{\mathbb{G}_0}) &\leq n_{\mathsf{C}} \cdot (\frac{q_{\mathsf{exec}}^2}{2^{|\mathsf{R}|}} + \mathsf{Adv}_G^{\mathsf{prf}}(\mathcal{A}')).
\end{aligned}$$

This concludes the proof.

Theorem 9. *[*$\mathsf{S.C.Imp}$*-resistance.] Let* $G : \{0,1\}^{\kappa} \times \{0,1\}^{d} \times \{0,1\}^{t} \times \{0,1\}^{t} \to \{0,1\}^{n}$ *be our specified function specified in section 4 and* Π *the* AKA *protocol specified in section 4. Consider a* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{s}}, q_{\mathsf{Op}}, q_G)$*-adversary* $\mathcal{A}$ *against the* $\mathsf{S.C.Imp}$*-security of the protocol* Π*, running in time* t *and creating at most*

q_{exec} *party instances with at most* q_{res} *resynchronizations per instance, corrupting at most* q_{s} *servers and making at most* q_{Op} $\mathsf{OpAccess}$ *queries per operator per corrupted server and making* q_G *queries to the function* G*. Denote the advantage of this adversary as* $\mathsf{Adv}_{\Pi}^{\mathsf{S.C.Imp}}(\mathcal{A})$*. Then there exists a* $(t' \approx O(t), q' = 5 \cdot q_{\mathsf{Op}} \cdot q_{\mathsf{s}} + q_G + q_{\mathsf{exec}}(q_{\mathsf{res}} + 2))$*-prf-adversary* $\mathcal{A}'$ *on* G *such that:*

$$\mathsf{Adv}_{\Pi}^{\mathsf{S.C.Imp}}(\mathcal{A}_{\mathbb{G}_0}) \leq n_{\mathsf{C}} \cdot \left(2 \cdot \mathsf{Adv}_{G}^{\mathsf{prf}}(\mathcal{A}') + \frac{(q_{\mathsf{exec}} + q_{\mathsf{s}} \cdot q_{\mathsf{Op}})^2}{2^{|\mathsf{R}|}} + \frac{q_{\mathsf{exec}} \cdot q_{\mathsf{res}}}{2^{|\mathsf{Res}|}} + \frac{1}{2^{\kappa}}\right).$$

Proof. **Game** $\mathbb{G}_0$: This game works as the $\mathsf{S.C.Imp}$-game: When the adversary $\mathcal{A}$ stops, it is said to win if there exists an instance S_i that ends in an accepting state with session and partner ID sid and pid such that: (a) pid is not adversarially controlled (its long-term key $\mathsf{sk}_{\mathsf{pid}}$ has not been corrupted), (b) no other instance C_i exists for $\mathsf{pid} = \mathsf{S}_i$ that ends in an accepting state, such that the both entities have the same session ID sid.

Game $\mathbb{G}_1$: This game works as the previous game $\mathbb{G}_0$ but including the new query $\mathsf{Corrupt}(\mathsf{P}, \mathtt{type})$, i.e with the presence of operator keys corruption (as detailed in the previous proof). The reduction from the game $\mathbb{G}_0$ to the game $\mathbb{G}_1$ is the as 6. As before, it holds that:

$$\Pr[\mathcal{A}_{\mathbb{G}_0}\mathsf{wins}] \leq \Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins}].$$

Game $\mathbb{G}_2$: We modify $\mathbb{G}_1$ to only interact with a single client (any future $\mathsf{CreateCl}$ calls for a client would be answered with an error symbol $\perp$). The challenger only generates a single operator key, which is associated with the operator chosen for the registered client. As indicated before, the security loss is given by:

$$\Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins}] \leq n_{\mathsf{C}} \cdot \Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}].$$

Game $\mathbb{G}_3$: We modify $\mathbb{G}_2$ to ensure that the random values sampled by any authentication challenge are always unique.

This gives us a security loss (related to the collisions between the R in two different instances) of

$$\left|\Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}]\right| \leq \frac{(q_{\mathsf{exec}} + q_{\mathsf{s}} \cdot q_{\mathsf{Op}})^2}{2^{|\mathsf{R}|}}.$$

Game $\mathbb{G}_4$: This game behaves as the game $\mathbb{G}_3$ with the restriction to only interact with only one server. The benefices lost is the ability to obtain some authentication challenges from corrupted servers. We recall that the challenge is split in five parts: a random value, a masked version of the fresh sequence number (an one-time-pad based on an anonymity key generated by the function G), two mac computed with the function G and both session keys. Moreover, we note that all the call of the function G take in input a specific value of the

related server, denoted $\mathsf{Id_S}$. Corrupted servers permit to obtain challenges based on the fresh sequence number but different random and server identifier values. So the related security loss is given by the collision on two outputs of the same function G with two different inputs (the only differences between both inputs are at least the value of the network identifier) and by the indistinguishability of the function G which are guaranteed by the pseudorandomness of G. We recall that the Test Phase of the game can be only focus on a fresh server which is or was never corrupted. This give us a security loss

$$\left| \Pr[\mathcal{A}_{\mathbb{G}_4}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}] \right| \leq \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}).$$

Game $\mathbb{G}_5$: We modify $\mathbb{G}_4$ to replace outputs to calls to all the internal cryptographic functions by truly random, but consistent values (they are independent of the input, but the same input gives the same output). As detailed in the key-secrecy, we obtain:

$$\left| \Pr[\mathcal{A}_{\mathbb{G}_4}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_5}\mathsf{wins}] \right| \leq \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}).$$

Winning $\mathbb{G}_5$: At this point, the adversary plays a game with a single client. A server instance S_i only accepts $\mathcal{A}_{\mathbb{G}_5}$, if this latter can generate a fresh an authentication response Res for some session sid. Assume that this happens against accepting instance S_i of the server, for some target session sid. Note that the value Res computed by C_i is purely random, but consistent. Thus, the adversary has three options for each of these values: (a) forwarding a value already received from the honest client for the same input values of which $\mathsf{sk_C}$ is unknown; (b) guessing the key $\mathsf{sk_C}$; or (c) guessing the value. The first option yields no result, since it implies there exists a previous client instance with the same session id sid as the client.

The second option happens with a probability of $2^{-|\mathsf{sk_C}|}$. The third option occurs with a probability of $2^{-|\mathsf{Res}|}$ per session (with or without resynchronization) per client, thus a total of $q_{\mathsf{exec}} \cdot 2^{-|\mathsf{Res}|}$. Thus,

$$\Pr[\mathcal{A}_{\mathbb{G}_5} wins] = 2^{-|\mathsf{sk_C}|} + q_{\mathsf{exec}} \cdot q_{\mathsf{res}} \cdot (2^{-|\mathsf{Res}|}).$$

Security statement: This yields the following result:

$$\mathsf{Adv}^{\mathsf{S.C.Imp}}_{\Pi}(\mathcal{A}_{\mathbb{G}_0}) \leq n_{\mathsf{C}} \cdot (2 \cdot \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}') + \frac{(q_{\mathsf{exec}} + q_{\mathsf{s}} \cdot q_{\mathsf{Op}})^2}{2^{|\mathsf{R}|}} + \frac{q_{\mathsf{exec}} \cdot q_{\mathsf{res}}}{2^{|\mathsf{Res}|}} + \frac{1}{2^{|\mathsf{sk_C}|}}).$$

Theorem 10. *[W.S.Imp-**resistance.**] Let $G : \{0,1\}^{\kappa} \times \{0,1\}^{d} \times \{0,1\}^{t} \times \{0,1\}^{t} \rightarrow \{0,1\}^{n}$ be our specified function specified in section 4 and Π our protocol specified in section 4. Consider a $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{G}})$-adversary $\mathcal{A}$ against the* W.S.Imp*-security of the protocol Π, running in time t, creating at most q_{exec} party instances, running at most q_{res} re-synchronizations per each instance, and making at most q_{G} queries to the function G. Denote the advantage of this adversary as $\mathsf{Adv}^{\mathsf{W.S.Imp}}_{\Pi}(\mathcal{A})$. Then there exists a $(t' \approx t, q = q_{\mathsf{exec}} \cdot (q_{\mathsf{res}} + 2) + q_{\mathsf{G}})$-adversary $\mathcal{A}$'*

with an advantage $\mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}')$ *of winning against the pseudorandomness of the function* G*, such that:*

$$\mathsf{Adv}^{\mathsf{W.S.Imp}}_{\Pi}(\mathcal{A}) \leq n_{\mathsf{C}} \cdot \left(\mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}') + \frac{q_{\mathsf{exec}} \cdot q_{\mathsf{res}}}{2^{|\mathsf{Mac_S}|}} + \frac{1}{2^{\kappa}} \right).$$

Proof. We prove this statement in three steps, similarly to the previous W.K.Ind proof. We recall that the adversary cannot corrupt the server.

Game $\mathbb{G}_0$: This game works as the S.Imp-game stipulated in section 3.

Game $\mathbb{G}_1$: This game works as the previous game $\mathbb{G}_0$ but including the new query $\mathsf{Corrupt}(\mathsf{P}, \mathtt{type})$. This game is the same as the game $\mathbb{G}_0$ in the proof of the weak key-indistinguishability theorem. As before, it holds that:

$$\Pr[\mathcal{A}_{\mathbb{G}_0}\mathsf{wins}] \leq \Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins}].$$

Note that adversary $\mathcal{A}_{\mathbb{G}_1}$ makes no extra query.

Game $\mathbb{G}_2$: We modify $\mathbb{G}_1$ to only allow interactions with a single client. The challenger generates only a single operator key, which is associated with the operator chosen for the registered client.

As indicated before the security loss is given by:

$$\Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins}] \leq n_{\mathsf{C}} \cdot \Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}].$$

Game $\mathbb{G}_3$: We modify $\mathbb{G}_2$ to replace outputs to calls to the function G by truly random, but consistent values (they are independent of the input, but the same input gives the same output). As before, it holds that:

$$\left| \Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}] \right| \leq \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}').$$

Winning $\mathbb{G}_3$: At this point, the adversary plays a game against a single client C, which only accepts $\mathcal{A}_{\mathbb{G}_3}$, if $\mathsf{Mac_S}$ is verified for some session sid. Assume that this happens against accepting instance C_i of the target client, for some target session sid. Note that the MAC value $\mathsf{Mac_S}$ computed by C_i is purely random, but consistent. Thus, the adversary has three options: (a) forwarding a value already received from the honest server for the same input values $\mathsf{R}, \mathsf{Sqn}, \mathsf{sk_{Op}}, \mathsf{sk_C}$, of which $\mathsf{sk_C}$ is unknown; (b) guessing the key $\mathsf{sk_C}$; or (c) guessing the vector. The former option yields no result, since it implies a server instance with the same session id sid as the client. The second option happens with a probability of $2^{-|\mathsf{sk_C}|}$. The third option occurs with a probability of $2^{-|\mathsf{Mac_S}|}$ per session (which is to say per instance and per re-synchronization), thus a total of $q_{\mathsf{exec}} \cdot q_{\mathsf{res}} 2^{-|\mathsf{Mac_S}|}$.

Thus,

$$\Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}] = 2^{-|\mathsf{sk_C}|} + q_{\mathsf{exec}} \cdot q_{\mathsf{res}} \cdot 2^{-|\mathsf{Mac_S}|}.$$

Security statement: This yields the following result:

$$\mathsf{Adv}^{\mathsf{W.S.Imp}}_{\Pi}(\mathcal{A}_{\mathbb{G}_0}) \leq n_{\mathsf{C}} \cdot \left(2^{-|\mathsf{sk_C}|} + q_{\mathsf{exec}} \cdot q_{\mathsf{res}} \cdot 2^{-|\mathsf{Mac_S}|} + \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}')\right).$$

Theorem 11. *[St.Conf-**resistance.**] Let G and G^* be our specified functions specified in section 4 and Π our fixed variant of the AKA protocol specified in section 4. Consider a $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{Op}}, q_G, q_{G^*})$-adversary $\mathcal{A}$ against the St.Conf-security of the protocol Π, running in time t and creating at most q_{exec} party instances with at most q_{res} resynchronizations per instance, making at most q_{Op} queries to oracle OpAccess and making q_G (resp. q_{G^*}) queries to the function G (resp. G^*). Denote the advantage of this adversary as $\mathsf{Adv}^{\mathsf{St.Conf}}_{\Pi}(\mathcal{A})$. Then there exist a $(t' \approx O(t), q' = q_G + q_{\mathsf{exec}}(5 + q_{\mathsf{res}}))$-prf-adversary $\mathcal{A}_1$ on G and $(t' \approx O(t), q' = q_{G^*})$-prf-adversary $\mathcal{A}_2$ on G^* a such that:*

$$\mathsf{Adv}^{\mathsf{St.Conf}}_{\Pi}(\mathcal{A}) \leq n_{\mathsf{C}} \cdot \left(\frac{1}{2^{|\mathsf{sk_C}|}} + \frac{1}{2^{|\mathsf{sk_{Op}}|}} + \frac{2}{2^{|\mathsf{Sqn}|}} + \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}_1) + \mathsf{Adv}^{\mathsf{prf}}_{G^*}(\mathcal{A}_2)\right).$$

Proof. Our proof has the following hops.

Game $\mathbb{G}_0$: This game works as the St.Conf-game stipulated in our security model. The goal of the adversary $\mathcal{A}_{\mathbb{G}_0}$ is to recover at least one secret value, i.e the subscriber key $\mathsf{sk_C}$, my operator key $\mathsf{sk_{Op}}$ or the subscriber sequence number $\mathsf{Sqn_C}$ for a fresh instance.

Game $\mathbb{G}_1$: We modify $\mathbb{G}_0$ to only allow interactions with one operator. The challenger related to the game $\mathbb{G}_1$ only generates a single operator key, which is associated with the operator chosen for the registered client. We proceed as follows: for any adversary $\mathcal{A}_{\mathbb{G}_0}$ winning the game $\mathbb{G}_0$ with a no-negligible success probability $\epsilon\mathcal{A}_{\mathbb{G}_0}$. we propose to construct an adversary $\mathcal{A}_{\mathbb{G}_1}$ winning the game $\mathbb{G}_1$ with a black-box access to the adversary $\mathcal{A}_{\mathbb{G}_0}$.

Adversary $\mathcal{A}_{\mathbb{G}_1}$ begins by choosing a single operator Op. It generates $n_{\mathsf{Op}} - 1$ operator keys, denoted $\mathsf{rsk_{Op^*}}$. Then, for every user registration request that $\mathcal{A}_{\mathbb{G}_0}$ sends to its challenger, $\mathcal{A}_{\mathbb{G}_1}$ responds as follows: if the request $\mathsf{CreateCl}(.)$ takes in input the operator Op, then it forwards the same query to its own oracle. Else, if $\mathcal{A}_{\mathbb{G}_0}$ sends a registration request based on any operator $\mathsf{Op}^* \neq \mathsf{Op}$, $\mathcal{A}_{\mathbb{G}_1}$ simulates the registration, generating a subscriber key $\mathsf{sk_{C^*}}$ and a sequence number $\mathsf{Sqn_{C^*}}$, returning the latter value. Moreover, each new client registered with the operator Op (resp. any Op^*) is associated with the related operator key $\mathsf{sk_{Op}}$(resp. $\mathsf{rsk_{Op^*}}$).

We distinguish between two types of clients: we denote C^* the clients which are registered with an operator $\mathsf{Op}^* \neq \mathsf{Op}$, and C the ones with the operator Op.

In the rest of the simulation, whenever $\mathcal{A}_{\mathbb{G}_0}$ makes a query to an instance of some party C^* (from any operator except Op), the adversary $\mathcal{A}_{\mathbb{G}_1}$ simulates the response using the values $\mathsf{sk_{C^*}}$, $\mathsf{rsk_{Op^*}}$, and the current value of $\mathsf{Sqn_{C^*}}$. For the other clients, the query is forwarded by $\mathcal{A}_{\mathbb{G}_1}$ to its own challenger.

Any corruption or reveal queries are dealt with in a similar way. Note that $\mathcal{A}_{\mathbb{G}_1}$ cannot query $\mathsf{Corrupt}$ to its adversary (this is a condition of freshness). The simulation is thus perfect up to the Test query.

In the Test query, $\mathcal{A}_{\mathbb{G}_0}$ chooses a $\mathsf{fresh\ instance}$ and sends it to $\mathcal{A}_{\mathbb{G}_1}$ (acting as a challenger). Note that $\mathcal{A}_{\mathbb{G}_1}$ will be able to test whether this instance is fresh, as freshness is defined in terms of $\mathcal{A}_{\mathbb{G}_0}$'s queries. If $\mathcal{A}_{\mathbb{G}_0}$ queries an instance C_i^* for the Test query ,then $\mathcal{A}_{\mathbb{G}_1}$ aborts the simulation, tests a random tuple about any fresh instance of the client C (creating one if necessary), winning with probability $\frac{1}{2^{|\mathsf{sk_C}|}} + \frac{1}{2^{|\mathsf{sk_{Op}}|}} + \frac{1}{2^{|\mathsf{Sqn_C}|}} + \frac{1}{2^{|\mathsf{Sqn_{Op,C}}|}}$. Else, if $\mathcal{A}_{\mathbb{G}_0}$ sends a tuple of a fresh instance of C_i, $\mathcal{A}_{\mathbb{G}_1}$ forwards this choice to its challenger and receives the challenger's output which contains the result of this game.

Denote by E_1 the event that adversary $\mathcal{A}_{\mathbb{G}_0}$ tests an instance C_i (from the chosen operator Op), while $\bar{\mathsf{E}_1}$ denotes the event that $\mathcal{A}_{\mathbb{G}_0}$ chooses to test C_i^*.

It holds that:

$$\begin{aligned}\Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins}] &= \Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins} \mid \mathsf{E}_1] \cdot \Pr[\mathsf{E}_1] + \\ &\quad \Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins} \mid \bar{\mathsf{E}_1}] \cdot \Pr[\bar{\mathsf{E}_1}] \\ &\geq \frac{1}{n_{\mathsf{Op}}} \Pr[\mathcal{A}_{\mathbb{G}_0}\mathsf{wins}] + \big(1 - \frac{1}{n_{\mathsf{Op}}}\big) \cdot \\ &\quad \big(\frac{1}{2^{|\mathsf{sk_C}|}} + \frac{1}{2^{|\mathsf{sk_{Op}}|}} + \frac{2}{2^{|\mathsf{Sqn}|}}\big).\end{aligned}$$

That implies:

$$\Pr[\mathcal{A}_{\mathbb{G}_0}\mathsf{wins}] \leq n_{\mathsf{Op}} \cdot \Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins}].$$

Game $\mathbb{G}_2$: We modify $\mathbb{G}_1$ to only allow interactions with a single client (any future $\mathsf{CreateCl(Op)}$ calls for a client would be answered with an error symbol $\perp$). We recall that the two adversaries $\mathcal{A}_{\mathbb{G}_1}$ and $\mathcal{A}_{\mathbb{G}_2}$ interact with clients from a single operator key, denoted Op, which is associated with the operator key $\mathsf{sk_{Op}}$. We proceed as follows: for any adversary $\mathcal{A}_{\mathbb{G}_1}$ winning the game $\mathbb{G}_2$ with a no-negligible success probability $\epsilon\mathcal{A}_{\mathbb{G}_1}$. we propose to construct an adversary $\mathcal{A}_{\mathbb{G}_2}$ winning the game $\mathbb{G}_2$ with a black-box access to the adversary $\mathcal{A}_{\mathbb{G}_1}$.

Adversary $\mathcal{A}_{\mathbb{G}_2}$ begins by choosing a single client C. For every user registration request that $\mathcal{A}_{\mathbb{G}_1}$ sends to its challenger, $\mathcal{A}_{\mathbb{G}_2}$ responds as follows: for a new client $\mathsf{C}^* \neq \mathsf{C}$ it generates $\mathsf{sk_{C^*}}$ and $\mathsf{Sqn_{C^*}}$, returning the latter value.

In the rest of the simulation, whenever $\mathcal{A}_{\mathbb{G}_1}$ makes a query to an instance of some party C^*, the adversary $\mathcal{A}_{\mathbb{G}_2}$ simulates the response using the oracle of the function G^* and the values $\mathsf{sk_{C^*}}$ and the current value of $\mathsf{Sqn_{C^*}}$. For the target client C, any queries are forwarded by $\mathcal{A}_{\mathbb{G}_2}$ to its challenger. Any corruption or reveal queries are dealt with in a similar way. Note that $\mathcal{A}_{\mathbb{G}_2}$ cannot query $\mathsf{Corrupt}$ to its adversary (this is a condition of freshness). The simulation is thus perfect up to the Test query.

In the Test query, $\mathcal{A}_{\mathbb{G}_1}$ chooses a $\mathsf{fresh\ instance}$ and sends it to $\mathcal{A}_{\mathbb{G}_2}$ (acting as a challenger). Note that $\mathcal{A}_{\mathbb{G}_2}$ will be able to test whether this instance is fresh, as freshness is defined in terms of $\mathcal{A}_{\mathbb{G}_1}$'s queries. If $\mathcal{A}_{\mathbb{G}_1}$ queries Test with a client other than the target client C, then $\mathcal{A}_{\mathbb{G}_2}$ aborts the simulation, tests a random

tuple as the previous reduction. Else, if $\mathcal{A}_{\mathbb{G}_1}$ queried a fresh instance of C, $\mathcal{A}_{\mathbb{G}_2}$ forwards this choice to its challenger and receives the challenger's which contains the result of this game. It holds that:

$$\Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins}] \leq \mathsf{n}_{\mathsf{C},\mathsf{Op}} \cdot \Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}].$$

, with at most $\mathsf{n}_{\mathsf{C},\mathsf{Op}}$ clients by operator.

Game $\mathbb{G}_3$: We modify $\mathbb{G}_2$ to replace outputs of the internal cryptographic functions by truly random, but consistent values (they are independent of the input, but the same input gives the same output). We argue that the security loss is precisely the advantage of the adversary $\mathcal{A}$ against the pseudorandomness of functions G and G^*. Note that the total number of queries to the related functions are at most $\mathsf{q}_G + q_{\mathsf{exec}}(5 + q_{\mathsf{res}})$ queries to the function G.

$$\left| \Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}] \right| \leq \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{prf}}_{G^*}(\mathcal{A}).$$

Winning Game $\mathbb{G}_3$: At this point, the adversary plays a game with an uncorruptible single client C_i in a protocol including truly but consistent values. She wins if she can output a tuple $(\mathsf{C}_i, \mathsf{sk}^*_{\mathsf{C}}, \mathsf{sk}^*_{\mathsf{Op}}, {\mathsf{Sqn}_{\mathsf{C}}}^*, {\mathsf{Sqn}_{\mathsf{Op},\mathsf{C}}}^*)$ such as at least one of these values corresponds to the real related secret value of the instance C_i. Thus, the adversary has only one choice to win this game: guessing each value. So the probability that the adversary $\mathcal{A}_{\mathbb{G}_3}$ wins is as follows:

$$\Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}] = \frac{1}{2^{|\mathsf{sk}_{\mathsf{C}}|}} + \frac{1}{2^{|\mathsf{sk}_{\mathsf{Op}}|}} + \frac{2}{2^{|\mathsf{Sqn}|}}.$$

Theorem 12. *[*Sound*-security.] Let $G : \{0,1\}^\kappa \times \{0,1\}^d \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^n$ be our specified function in section 4 and Π the protocol specified in section 4. Consider a $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{Op}}, q_{\mathsf{G}}, \epsilon)$-server-sound-adversary $\mathcal{A}$ against the soundness of the protocol Π, running in time t, creating at most q_{exec} party instances with at most q_{res} resynchronizations per instance, making at most q_{Op} queries to any operator Op and at most q_{G} queries to the function G. Denote the advantage of this adversary as $\mathsf{Adv}^{\mathsf{Sound}}_{\Pi}(\mathcal{A})$. Then there exist a $(t' \approx t, q' = 5 \cdot q_{\mathsf{Op}} + q_G + q_{\mathsf{exec}}(2 + q_{\mathsf{res}}))$-adversary $\mathcal{A}'$ with an advantage $\mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}')$ of winning against the pseudorandomness of the function G, such that:*

$$\mathsf{Adv}^{\mathsf{Sound}}_{\Pi}(\mathcal{A}) \leq n_{\mathsf{C}} \cdot \left(2 \cdot \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}') + \frac{q_{\mathsf{exec}} \cdot q_{\mathsf{res}}}{2^{|\mathsf{Mac_S}|+|\mathsf{Sqn}|}} + \frac{1}{2^\kappa} \right).$$

Proof. **Game** $\mathbb{G}_0$: This game works as the game Sound-game stipulated in our security model. The goal of this adversary $\mathcal{A}_{\mathbb{G}_0}$ is similar as the $\mathsf{S.Imp}$-game but with a different adversary; indeed in the $\mathsf{S.Imp}$-game is a MiM adversary and in the Sound-game, we have a *legitimate-but-malicious* server-adversary.

Game $\mathbb{G}_1$: We consider the game $\mathbb{G}_1$ as the $\mathsf{S.S.Imp}$-game (as previously detailed) but including the specific query $\mathsf{Corrupt}(\mathsf{P}, \mathtt{type})$, i.e with the presence of operator keys corruption. We have used a such query in some previous security proofs. We proceed to show that, for any adversary $\mathcal{A}_{\mathbb{G}_0}$ winning the game $\mathbb{G}_0$ with an advantage $\mathsf{Adv}^{\mathsf{Sound}}_{\Pi}(\mathcal{A}_{\mathbb{G}_0})$, there exists an adversary $\mathcal{A}_{\mathbb{G}_1}$ with black-box

access to the adversary $\mathcal{A}_{\mathbb{G}_0}$ wins game $\mathbb{G}_1$. Both adversaries play her related game with oracles. The following oracles are similar in the two games: Send, CreateCl, Init, Execute, Reveal, and StReveal. So for each query related to these oracles from the adversary $\mathcal{A}_{\mathbb{G}_0}$, the adversary $\mathcal{A}_{\mathbb{G}_1}$ forwards these queries to its own challenger and sends to $\mathcal{A}_{\mathbb{G}_0}$ the related answers. Now focus on the two last oracles which can be used by the adversary $\mathcal{A}_{\mathbb{G}_0}$: OpAccess and Corrupt.

At first, we recall that the OpAccess in the game $\mathbb{G}_0$ takes in input a client identifier and outputs, for our protocol, an authentication vector composed by the tuple $\mathsf{AV} = (\mathsf{R}, \mathsf{Autn}, \mathsf{Mac}_{\mathsf{C}}, \mathsf{CK}, \mathsf{IK})$. To simulate the answer of the oracle $\mathsf{OpAccess}(\mathsf{C}_i)$, the $\mathcal{A}_{\mathbb{G}_1}$ uses the query $\mathsf{Execute}(S, \mathsf{C}_i)$ (with the server related to the *legitimate-but-malicious* adversary) and $\mathsf{Reveal}(\mathsf{C}, i)$.

Now, focus on the simulation of the Corrupt answer. We recall that we have two possible inputs: a client or an operator. In the Corrupt oracle takes in input a client, the adversary $\mathcal{A}_{\mathbb{G}_1}$ uses its own Corrupt oracle to obtain the related answer. If the input is an operator, $\mathcal{A}_{\mathbb{G}_1}$ needs to forge the following values: the operator key $\mathsf{sk}_{\mathsf{Op}}$, and for each client of this operator the tuple $(\mathsf{UID}, \mathsf{sk}_{\mathsf{UID}}, \mathsf{st}_{\mathsf{Op},\mathsf{C}})$. To simulate a such answer, $\mathcal{A}_{\mathbb{G}_1}$ uses its specific $\mathsf{Corrupt}(\mathsf{C})$ and $\mathsf{StReveal}(\mathsf{C}, i, 1)$ for each client Cof this operator.

So at this point, the adversary $\mathcal{A}_{\mathbb{G}_1}$ can simulate any query from the adversary $\mathcal{A}_{\mathbb{G}_0}$. At the end of the simulation, the adversary $\mathcal{A}_{\mathbb{G}_1}$ replays the impersonation's attempt from the adversary $\mathcal{A}_{\mathbb{G}_0}$. Thus, we have:

$$\Pr[\mathcal{A}_{\mathbb{G}_0}\mathsf{wins}] = \Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins}].$$

Winning game $\mathbb{G}_1$: This game follows the game $\mathbb{G}_1$ described in the reduction proof of the theorem S.S.Imp. Thus, we have :

$$\mathsf{Adv}^{\mathsf{S.S.Imp}}_{\Pi}(\mathcal{A}_{\mathbb{G}_1}) \leq n_{\mathsf{C}} \cdot (2 \cdot \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}') + \frac{q_{\mathsf{exec}} \cdot q_{\mathsf{res}}}{2^{|\mathsf{Mac_S}|}} + \frac{1}{2^{\kappa}}).$$

Theorem 13. *[S.K.Ind-resistance.] Let $G : \{0,1\}^{\kappa} \times \{0,1\}^{d} \times \{0,1\}^{t} \times \{0,1\}^{t} \rightarrow \{0,1\}^{n}$ be our specified function specified in section 4 and Π the fixed AKA protocol specified in section 4. Consider a $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{s}}, q_{\mathsf{Op}}, q_G)$-adversary $\mathcal{A}$ against the S.K.Ind-security of the protocol Π, running in time t and creating at most q_{exec} party instances with at most q_{res} resynchronizations per instance, corrupting at most q_{s} servers and making at most q_{Op} OpAccess queries per operator per corrupted server and making q_G queries to the function G. Denote the advantage of this adversary as $\mathsf{Adv}^{\mathsf{S.K.Ind}}_{\Pi}(\mathcal{A})$. Then there exists a $(t' \approx O(t), q' = 5 \cdot q_{\mathsf{s}} \cdot q_{\mathsf{Op}} + q_G + q_{\mathsf{exec}}(q_{\mathsf{res}} + 2))$-prf-adversary $\mathcal{A}'$ on G such that:*

$$\mathsf{Adv}^{\mathsf{S.K.Ind}}_{\Pi}(\mathcal{A}) \leq n_{\mathsf{C}} \cdot \left(\frac{(q_{\mathsf{exec}} + q_{\mathsf{s}} \cdot q_{\mathsf{Op}})^2}{2^{|\mathsf{R}|}} + 2 \cdot \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}') \right).$$

Proof. Our proof has the following hops.

Game $\mathbb{G}_0$: This game works as the S.K.Ind-game stipulated in our security model 3.

Game $\mathbb{G}_1$: We modify $\mathbb{G}_0$ to only consider the new query $\mathsf{Corrupt}(\mathsf{P}, \mathtt{type})$ but keeping the same goal. We note that this new query permits to consider the corruption of the key operator independently to the corruption of the subscriber keys. This new query behaves as follows:

$\mathsf{Corrupt}(\mathsf{P}, \mathtt{type})$: yields to the adversary the long-term keys of party $\mathsf{P} \neq \mathsf{S}$ (else, if the oracle takes as input $\mathsf{P} = \mathsf{S}$, then it behaves as usual calling the oracle **OpAccess**). The output of the oracle depends on the value $\mathtt{type} \in \{\mathtt{sub}, \mathtt{op}, \mathsf{all}\}$. If $\mathtt{type} = \mathtt{sub}$, then the returned value is sk_P. If $\mathtt{type} = \mathtt{op}$, then the oracle returns $\mathsf{sk}_{\mathsf{Op}}$. Then, for $\mathtt{type} = \mathsf{all}$, we return the both values sk_P, $\mathsf{sk}_{\mathsf{Op}}$. If $\mathtt{type} \in \{\mathtt{sub}, \mathsf{all}\}$, then P (and all its instances, past, present, or future), are considered to be adversarially controlled.

We argue that given any adversary $\mathcal{A}$ playing the game $\mathbb{G}_1$ and winning w.p. $\epsilon_\mathcal{A}$, the same adversary wins the game $\mathbb{G}_0$ w.p. at least $\epsilon_\mathcal{A}$ (this is trivial since in game $\mathbb{G}_1$, $\mathcal{A}$ has more information).

$$\Pr[\mathcal{A} \text{ wins } \mathbb{G}_0] \leq \Pr[\mathcal{A} \text{ wins } \mathbb{G}_1].$$

Game $\mathbb{G}_2$: We modify $\mathbb{G}_1$ to only allow interactions with a single client. The challenger generates only a single operator key, which is associated with the operator chosen for the registered client. As indicated before, the security loss is given by:

$$\Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}] \geq \frac{1}{n_\mathsf{C}} \Pr[\mathcal{A}_{\mathbb{G}_0}\mathsf{wins}] + \frac{1}{2} \cdot \big(1 - \frac{1}{n_\mathsf{C}}\big).$$

Game $\mathbb{G}_3$: We modify $\mathbb{G}_2$ to ensure that the random value sampled by honest server instances is always unique.

This gives us a security loss (related to the respective collisions between the R in two different instances) of

$$\big|\Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}]\big| \leq \frac{(q_{\mathsf{exec}} + q_\mathsf{s} \cdot q_{\mathsf{Op}})^2}{2^{|\mathsf{R}|}}.$$

Game $\mathbb{G}_4$: This game behaves as the game $\mathbb{G}_3$ with the restriction to only interact with only one server. The benefices loss is the ability to obtain some authentication challenges from uncorrupted servers. Such authentication challenges can be either to give information about the used sequence number and the long term keys or to forge a fresh challenge replaying some parts of these challenges. We recall that the challenge is split in five parts: a random value, a masked version of the fresh sequence number (an one-time-pad based on an anonymity key generated by the function G), two mac computed with the function G and both session keys. Moreover, we note that all the call of the function G takes in input a specific value of the related server Id_S. Thus, the two

session keys can not directly reuse since the random value Rand is never reuse (see previous reduction). So, except when we obtain a collision, the session keys will be always different in each session.

So the related security loss is given by the collision on two outputs of the same function G with two different inputs (the only differences between the both inputs are at least the value of the network identifier) and by the indistinguishability of the function G which are both guaranteed by the pseudorandomness of G. We recall that the Test Phase of the game can be only focus on a network which is or was never corrupted. This give us a security loss

$$\left| \Pr[\mathcal{A}_{\mathbb{G}_4}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}] \right| \leq \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}).$$

Game $\mathbb{G}_5$: We modify $\mathbb{G}_4$ to replace outputs of the internal cryptographic functions by truly random, but consistent values (they are independent of the input, but the same input gives the same output). As indicated before, the security loss is given by:

$$\left| \Pr[\mathcal{A}_{\mathbb{G}_4}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_5}\mathsf{wins}] \right| \leq \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}).$$

Winning $\mathbb{G}_5$: At this point, the adversary plays a game in the presence of a single client C. The goal of this adversary is to distinguish a random session key to a fresh session key. But, in game $\mathbb{G}_5$, queries to G return truly random, consistent values. In this case, the adversary can do no better than guessing. Thus, we have:

$$\Pr[\mathcal{A}_{\mathbb{G}_5}\mathsf{wins}] = \frac{1}{2}.$$

Security statement: This yields the following result:

$$\mathsf{Adv}^{\mathsf{S.K.Ind}}_{\Pi}(\mathcal{A}_{\mathbb{G}_0}) \leq n_{\mathsf{C}} \cdot \left(\frac{(q_{\mathsf{exec}} + q_{\mathsf{s}} \cdot q_{\mathsf{Op}})^2}{2^{|\mathsf{R}|}} + 2 \cdot \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}) \right).$$

This concludes the proof.

Theorem 14. *[S.S.Imp-**resistance.**] Let $G : \{0,1\}^{\kappa} \times \{0,1\}^{d} \times \{0,1\}^{t} \times \{0,1\}^{t} \to \{0,1\}^{n}$ be our specified function specified in section 4 and Π our fixed variant of the AKA protocol specified in section 4. Consider a $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{s}}, q_{\mathsf{Op}}, q_{\mathsf{G}}, \epsilon)$-adversary $\mathcal{A}$ against the S.S.Imp-security of the protocol Π, running in time t and creating at most q_{exec} party instances with at most q_{res} resynchronizations per instance, corrupting at most q_{s} servers and making at most q_{Op} OpAccess queries per operator per corrupted server and making q_G queries to the function G. Denote the advantage of this adversary as $\mathsf{Adv}^{\mathsf{S.S.Imp}}_{\Pi}(\mathcal{A})$. Then there exists a $(t' \approx O(t), q' = 5 \cdot q_{\mathsf{s}} \cdot q_{\mathsf{Op}} + q_G + q_{\mathsf{exec}}(2 + q_{\mathsf{res}}))$-prf-adversary $\mathcal{A}'$ on G such that:*

such that:

$$\mathsf{Adv}^{\mathsf{S.S.Imp}}_{\Pi}(\mathcal{A}_{\mathbb{G}_0}) \leq n_{\mathsf{C}} \cdot \left(\frac{q_{\mathsf{exec}} + q_{\mathsf{res}}}{2^{|\mathsf{Macs}|}} + \frac{1}{2^{\kappa}} + 2 \cdot \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}') \right).$$

Proof. **Game** $\mathbb{G}_0$: This game works as the S.S.Imp-game detailed in the section 3.

Game $\mathbb{G}_1$: This game works as the previous game $\mathbb{G}_0$ but including the new query $\mathsf{Corrupt}(\mathsf{P}, \mathtt{type})$, i.e with the presence of operator keys corruption. The reduction from the game $\mathbb{G}_0$ to the game $\mathbb{G}_1$ is the same as the security proof of the theorem 6. As before, it holds that:

$$\Pr[\mathcal{A}_{\mathbb{G}_0}\mathsf{wins}] \leq \Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins}].$$

Note that adversary $\mathcal{A}_{\mathbb{G}_1}$ makes no extra query.

Game $\mathbb{G}_2$: We modify $\mathbb{G}_1$ to only interact with a single client (any future CreateCl calls would be answered with an error symbol $\perp$). The challenger only generates a single operator key, which is associated with the operator chosen for the registered client. As indicated before, the security loss is given by:

$$\Pr[\mathcal{A}_{\mathbb{G}_1}\ \mathsf{wins}] \leq n_{\mathsf{C}} \cdot \Pr[\mathcal{A}_{\mathbb{G}_2}\ \mathsf{wins}].$$

Game $\mathbb{G}_3$: This game behaves as the game $\mathbb{G}_2$ with the restriction to only interact with only one server. The benefices loss is the ability to obtain some authentication challenges from uncorrupted servers. As detailed in the proof of the strong key-indistinguishability, the related security loss is given by the pseudorandomness of the function G. We recall that the Test Phase of the game can be only focus on a network which is or was never corrupted. This give us a security loss

$$\left| \Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}] \right| \leq \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}).$$

Game $\mathbb{G}_4$: We modify $\mathbb{G}_3$ to replace outputs to calls to all the internal cryptographic functions by truly random, but consistent values (they are independent of the input, but the same input gives the same output). As detailed in the key-secrecy, we obtain:

$$\left| \Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_4}\mathsf{wins}] \right| \leq \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}).$$

Winning $\mathbb{G}_4$: At this point, the adversary plays a game with a single client C_i, which only accepts $\mathcal{A}_{\mathbb{G}_4}$, if the authentication challenge is verified for some session sid. Assume that this happens against accepting instance $\mathsf{C_i}$ of the target client, for some target session sid. Note that the MAC value $\mathsf{Mac_s}$ computed by C_i is purely random, but consistent. Thus, the adversary has three options: (a) forwarding a value already received from a honest server for the

same input values R; $\mathsf{Id_S}$; Sqn; $\mathsf{sk_{Op}}$; $\mathsf{sk_C}$, of which $\mathsf{sk_C}$ is unknown; (b) guessing the key $\mathsf{sk_C}$; or (c) guessing the response. The first option yields no result since there are no collision between the transcript of two different servers since all the servers have a different server identifier $\mathsf{Id_S}$. The second option happens with a probability of $2^{-|\mathsf{sk_C}|}$. The third option occurs with a probability of $2^{-|\mathsf{Macs}|}$ per session (which is to say per instance and per re-synchronization), thus a total of $q_{\mathsf{exec}} \cdot q_{\mathsf{res}} \cdot 2^{-|\mathsf{Macs}|}$. Thus,

$$\Pr[\mathcal{A}_{\mathbb{G}_4} wins] = 2^{-|\mathsf{sk_C}|} + q_{\mathsf{exec}} \cdot q_{\mathsf{res}} \cdot 2^{-|\mathsf{Macs}|}.$$

Security statement: This yields the following result:

$$\mathsf{Adv}^{\mathsf{S.S.Imp}}_{\Pi}(\mathcal{A}_{\mathbb{G}_0}) \leq n_{\mathsf{C}} \cdot \left(\frac{q_{\mathsf{exec}} \cdot q_{\mathsf{res}}}{2^{|\mathsf{Macs}|}} + \frac{1}{2^{\kappa}} + 2 \cdot \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}') \right).$$

E Updated TUAK and MILENAGE

In our variant, we modified the inputs of the internal cryptographic algorithms to include the new value $\mathsf{Id_S}$. Thus, we need to provide an update of these algorithms. As specified previously, the AKA protocol can be based on two different sets of algorithms: TUAK and MILENAGE. To preserve backwards compatibility, we propose to keep and update these two sets.

The seven internal cryptographic functions used in the AKA protocol takes in inputs the following values:

- keys: the couple of 128-bit (or 256-bit) keys: the subscriber key $\mathsf{sk_C}$ and the operator key $\mathsf{sk_{Op}}$.
- Sqn (for the functions $\mathbf{Upd\_F}_1$ and $\mathbf{Upd\_F}_1^*$): a 48-bit sequence number.
- AMF (except for the functions $\mathbf{Upd\_F}_1$ and $\mathbf{Upd\_F}_1^*$): a 16-bit authentication field management.
- R: a 128-bit random value.
- $\mathsf{Id_S}$: a 128-bit (public) value characterizing the visited network.

We note that the functions $\mathbf{Upd\_F}_1$ and $\mathbf{Upd\_F}_1^*$ behave differently because they consider the sequence number in inputs.

Update of the MILENAGE **algorithms:** MILENAGE is the original set of algorithms which is currently implemented as detailed the specification 35.206 [1].

In order to ensure a stronger degree of security, we also modify the MILENAGE algorithms to output 128-bit MAC and session keys CK and IK.

Based on the Advanced Encryption Standard (AES), these functions compute firstly both values Top_C and Temp as follows:

$$\mathsf{Top}_C = \mathsf{sk_{Op}} \oplus \mathsf{AES}_{\mathsf{sk_C}}(\mathsf{sk_{Op}}), \mathsf{Temp} = \mathsf{AES}_{\mathsf{sk_C}}(\mathsf{R} \oplus \mathsf{Top}_C \oplus \mathsf{Id_S}).$$

The outputs of the MILENAGE algorithms are computed as follows:

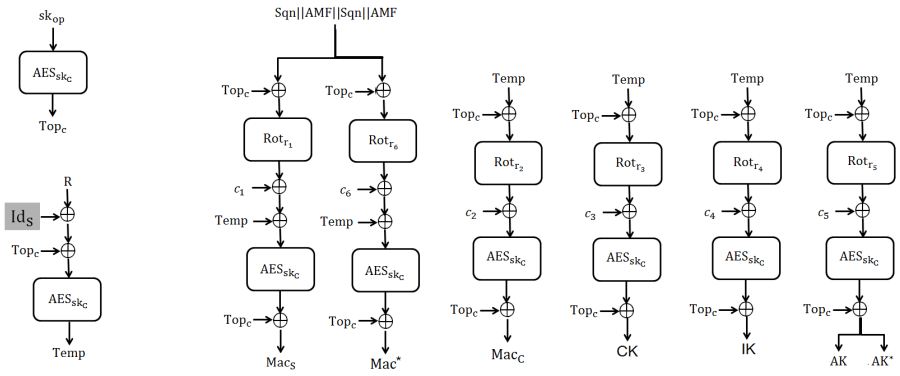


Fig. 10. Updated MILENAGE.

- **Output Upd_F$_1$:** $\mathsf{Mac_C} = \mathsf{AES_{sk_C}}(\mathsf{Temp} \oplus \mathsf{Rot_{r_1}}(\mathsf{Sqn}\|\mathsf{AMF}\|\mathsf{Sqn}\|\mathsf{AMF}) \oplus \mathsf{c_1}) \oplus \mathsf{Top}_C$,
- **Output Upd_F$_1^*$:** $\mathsf{Mac}^* = \mathsf{AES_{sk_C}}(\mathsf{Temp} \oplus \mathsf{Rot_{r_6}}(\mathsf{Sqn}\|\mathsf{AMF}\|\mathsf{Sqn}\|\mathsf{AMF}) \oplus \mathsf{c_6}) \oplus \mathsf{Top}_C$,
- **Output Upd_F$_2$:** $\mathsf{Mac_S} = \mathsf{AES_{sk_C}}(\mathsf{Rot_{r_2}}(\mathsf{Temp} \oplus \mathsf{Top}_C) \oplus \mathsf{c_2}) \oplus \mathsf{Top}_C$
- **Output Upd_F$_3$:** $\mathsf{CK} = \mathsf{AES_{sk_C}}(\mathsf{Rot_{r_3}}(\mathsf{Temp} \oplus \mathsf{Top}_C) \oplus \mathsf{c_3}) \oplus \mathsf{Top}_C$,
- **Output Upd_F$_4$:** $\mathsf{IK} = \mathsf{AES_{sk_C}}(\mathsf{Rot_{r_4}}(\mathsf{Temp} \oplus \mathsf{Top}_C) \oplus \mathsf{c_4}) \oplus \mathsf{Top}_C$,
- **Output Upd_F$_5$:** $\mathsf{AK} = \lfloor \mathsf{AES_{sk_C}}(\mathsf{Rot_{r_5}}(\mathsf{Temp} \oplus \mathsf{Top}_C, \mathsf{r_5}) \oplus \mathsf{c_5}) \oplus \mathsf{Top}_C \rfloor_{0..47}$,
- **Output Upd_F$_5^*$:** $\mathsf{AK}^* = \lfloor \mathsf{AES_{sk_C}}(\mathsf{Rot_{r_5}}(\mathsf{Temp} \oplus \mathsf{Top}_C, \mathsf{r_5}) \oplus \mathsf{c_5}) \oplus \mathsf{Top}_C \rfloor_{80..127}$,

with the five integers $\mathsf{r_1} = 0$, $\mathsf{r_2} = 16$, $\mathsf{r_3} = 32$, $\mathsf{r_4} = 64$, $\mathsf{r_5} = 80$ and $\mathsf{r_6} = 96$ in the range $\{0, 127\}$, which define the number of positions the intermediate variables are cyclically rotated by the right, and the five 128-bit constants c_i such as:

- $\mathsf{c_1}[i] = 0, \forall i \in \{0, 127\}$.
- $\mathsf{c_2}[i] = 0, \forall i \in \{0, 127\}$, except that $\mathsf{c_2}[127] = 1$.
- $\mathsf{c_3}[i] = 0, \forall i \in \{0, 127\}$, except that $\mathsf{c_3}[126] = 1$.
- $\mathsf{c_4}[i] = 0, \forall i \in \{0, 127\}$, except that $\mathsf{c_4}[125] = 1$.
- $\mathsf{c_5}[i] = 0, \forall i \in \{0, 127\}$, except that $\mathsf{c_5}[124] = 1$.
- $\mathsf{c_6}[i] = 0, \forall i \in \{0, 127\}$, except that $\mathsf{c_6}[123] = 1$.

This is also described in Figure 10.

Update of the TUAK algorithms: TUAK is an alternative set of algorithms to MILENAGE based on the internal permutation of Keccak [12]. The specification TS 35.231 [2] details the internal algorithms of this set. We update these algorithms by only modifying the inputs of the second permutation. We recall that in this instantiation, the functions **Upd_F$_1^*$** and **Upd_F$_5^*$**, used for the resynchronization procedure, behave in the same way but use different values Inst', Inst.

We first compute the value Top_C as follows:

$$\mathsf{Top}_C = \lfloor \mathsf{f}_{\mathsf{Keccak}}(\mathsf{sk}_{\mathsf{Op}} \| \mathsf{Inst} \ \| \mathsf{AN} \| 0^{192} \| \mathsf{Key} \| \mathsf{Pad} \| 1 \| 0^{512}) \rfloor_{1..256}.$$

We note that the values AN, Inst', Inst, Pad are the same as used in the original TUAK algorithms and Key the (padded) subscriber key.

At this point, the behavior of the functions $\mathbf{Upd}\_\mathsf{F}_1$ (resp. $\mathbf{Upd}\_\mathsf{F}_1^*$) diverges from the other functions. Generating the related output, we compute the value Val_1 and for the others ones, we compute the value Val_2 which differ including the 128-bit value Id_S and the smaller paging value Pad.

$$\mathsf{Val}_1 = \mathsf{f}_{\mathsf{Keccak}}(\mathsf{Top}_C \| \mathsf{Inst}' \| \mathsf{AN} \| \mathsf{R} \| 0^{64} \| \mathsf{Key} \| \mathsf{Id}_\mathsf{S} \| \mathsf{Pad} \| 10^{512}),$$
$$\mathsf{Val}_2 = \mathsf{f}_{\mathsf{Keccak}}(\mathsf{Top}_C \| \mathsf{Inst}' \| \mathsf{AN} \| \mathsf{R} \| \mathsf{AMF} \| \mathsf{Sqn} \| \mathsf{Key} \| \mathsf{Id}_\mathsf{S} \| \mathsf{Pad} \| 10^{512}).$$

Then, we obtain the output of the seven functions truncating the related value as follows:

- **Output $\mathbf{Upd}\_\mathsf{F}_1$:** $\mathsf{Mac}_\mathsf{S} = \lfloor \mathsf{Val}_2 \rfloor_{0..127}$,
- **Output $\mathbf{Upd}\_\mathsf{F}_2$:** $\mathsf{Mac}_\mathsf{C} = \lfloor \mathsf{Val}_1 \rfloor_{0..127}$,
- **Output $\mathbf{Upd}\_\mathsf{F}_3$:** $\mathsf{CK} = \lfloor \mathsf{Val}_1 \rfloor_{256..383}$,
- **Output $\mathbf{Upd}\_\mathsf{F}_4$:** $\mathsf{IK} = \lfloor \mathsf{Val}_1 \rfloor_{512..639}$,
- **Output $\mathbf{Upd}\_\mathsf{F}_5$:** $\mathsf{AK} = \lfloor \mathsf{Val}_1 \rfloor_{768..815}$.

This is also depicted in Figure 11.

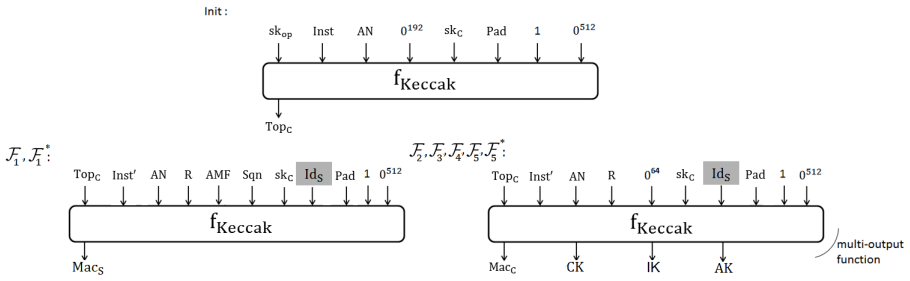


Fig. 11. Updated TUAK.

We note that the multi-output property is, as in the original version, not an issue for the security of the master key, since during one session we can have as many as four calls to the same function with similar inputs (and a different truncation).

F Security of TUAK and MILENAGE

In this section, we prove the pseudorandomness of both versions (classic and updated) of TUAK and MILENAGE algorithms.

The security of (classic and updated) TUAK. In order to prove the prf-security of the (classic and updated) TUAK algorithms, we assume that the truncated keyed internal Keccak permutation is a good pseudorandom function. We propose two generic constructions to model the TUAK algorithms: a first one, denoted G_{tuak} when the secret is based on the subscriber key $\mathsf{sk_C}$ and a second one, denoted G^*_{tuak} when is only based on the operator key.

It is worth noting that the construction of the TUAK functions is reminiscent of the Merkle-Damgård construction, where the output of the function f is an input of the next iteration of the function f. This is in contradiction with the Sponge construction used in the hash function Keccak given the internal permutation $\mathsf{f_{Keccak}}$. So we precise that this security proof does not directly imply an innovation on the Keccak construction.

We model the truncated keyed internal permutation of Keccak by the function f and f^*:

$$f(K, x\|y, i, j) = \lfloor \mathsf{f_{Keccak}}(x\|K\|y) \rfloor_{i..j},$$
$$f^*(K^*, x^*\|y^*, i, j) = \lfloor \mathsf{f_{Keccak}}(K^*\|x^*\|y^*) \rfloor_{i..j},$$

with $x \in \{0,1\}^{512}$, $K, K^* \in \{0,1\}^{\kappa}$, $y \in \{0,1\}^{1088-\kappa}$, $x^* \in \{0,1\}^{512+\kappa}, y^* \in \{0,1\}^{1088}$ and $i, j \in \{0,1\}^t$ with $\log_2(t-1) < 1600 \leq \log_2(t)$. We note that $\forall K, x, x^*, y, y^*, i, j$ such as $x = K^*\|x^*$ and $y^* = K\|y$, we have $f(K, x\|y, i, j) = f^*(K^*, x^*\|y^*, i, j)$. The input x (resp. x^*) can be viewed as the chaining variable of the cascade construction of G_{tuak} given f (resp. f^*), y (resp. y^*) is an auxiliary input of the function, and i and j define the size of the truncation. The construction G_{tuak} and G^*_{tuak} act as a generalization of the specific TUAK algorithms:

$$\begin{aligned}
\mathcal{F}_1(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Sqn}, \mathsf{AMF}) &= G_{\mathsf{tuak}}(\mathsf{sk_C}, \mathsf{inp}_1, 0, 127) = G^*_{\mathsf{tuak}}(\mathsf{sk_{Op}}, {\mathsf{inp^*}}_1, 0, 127),\\
\mathcal{F}^*_1(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Sqn}, \mathsf{AMF}) &= G_{\mathsf{tuak}}(\mathsf{sk_C}, \mathsf{inp}_2, 0, 127) = G^*_{\mathsf{tuak}}(\mathsf{sk_{Op}}, \mathsf{inp}^*_2, 0, 127),\\
\mathcal{F}_2(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R}) &= G_{\mathsf{tuak}}(\mathsf{sk_C}, \mathsf{inp}_3, 0, 127) = G^*_{\mathsf{tuak}}(\mathsf{sk_{Op}}, \mathsf{inp}^*_3, 0, 127),\\
\mathcal{F}_3(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R}) &= G_{\mathsf{tuak}}(\mathsf{sk_C}, \mathsf{inp}_3, 256, 383) = G^*_{\mathsf{tuak}}(\mathsf{sk_{Op}}, \mathsf{inp}^*_3, 256, 383),\\
\mathcal{F}_4(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R}) &= G_{\mathsf{tuak}}(\mathsf{sk_C}, \mathsf{inp}_3, 512, 639) = G^*_{\mathsf{tuak}}(\mathsf{sk_{Op}}, \mathsf{inp}^*_3, 512, 639),\\
\mathcal{F}_5(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R}) &= G_{\mathsf{tuak}}(\mathsf{sk_C}, \mathsf{inp}_3, 768, 815) = G^*_{\mathsf{tuak}}(\mathsf{sk_{Op}}, \mathsf{inp}^*_3, 768, 815),\\
\mathcal{F}^*_5(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R}) &= G_{\mathsf{tuak}}(\mathsf{sk_C}, \mathsf{inp}_4, 768, 815) = G^*_{\mathsf{tuak}}(\mathsf{sk_{Op}}, \mathsf{inp}^*_4, 768, 815),
\end{aligned}$$

with:

$\mathsf{inp}_1 = \mathsf{sk_{Op}}\|\mathsf{cst}_1\|\mathsf{cst}_5$,, $\mathsf{inp}_2 = \mathsf{sk_{Op}}\|\mathsf{cst}_1\|\mathsf{cst}_5$,
$\mathsf{inp}_3 = \mathsf{sk_{Op}}\|\mathsf{cst}_3\|\mathsf{cst}_5$,, $\mathsf{inp}_4 = \mathsf{sk_{Op}}\|\mathsf{cst}_4\|\mathsf{cst}_5$,
$\mathsf{inp}^*_1 = \mathsf{cst}_1\|\mathsf{keys}\|\mathsf{cst}_5$,, $\mathsf{inp}^*_2 = \mathsf{cst}_1\|\mathsf{keys}\|\mathsf{cst}_5$,
$\mathsf{inp}^*_3 = \mathsf{cst}_3\|\mathsf{keys}\|\mathsf{cst}_5$,, $\mathsf{inp}^*_4 = \mathsf{cst}_4\|\mathsf{keys}\|\mathsf{cst}_5$,
$\mathsf{cst}_1 = \mathsf{Inst}\|\mathsf{AN}\|0^{192}\|(\mathsf{Inst}'\|\mathsf{AN}\|\mathsf{R}\|\mathsf{AMF}\|\mathsf{Sqn})$,
$\mathsf{cst}_2 = \mathsf{Inst}\|\mathsf{AN}\|0^{192}\|(\mathsf{Inst}^{(2)}\|\mathsf{AN}\|\mathsf{R}\|\mathsf{AMF}\|\mathsf{Sqn})$,
$\mathsf{cst}_3 = \mathsf{Inst}\|\mathsf{AN}\|0^{192}\|(\mathsf{Inst}'\|\mathsf{AN}\|\mathsf{R}\|0^{64})$,
$\mathsf{cst}_4 = \mathsf{Inst}\|\mathsf{AN}\|0^{192}\|(\mathsf{Inst}^{(3)}\|\mathsf{AN}\|\mathsf{R}\|0^{64})$,
$\mathsf{cst}_5 = \mathsf{Pad}\|1\|0^{192}$,

We define the cascade construction G_{tuak} based on the function f as follows:

$$G_{\mathsf{tuak}}(K, \mathsf{val}, i, j) = f(K, f(K, \mathsf{val}_1 \| \mathsf{val}_3, 0, 256) \| \mathsf{val}_2 \| \mathsf{val}_3, i, j),$$
$$G^*_{\mathsf{tuak}}(K^*, val^*, i, j) = f^*(f^*, \mathsf{val}^*_1 \| \mathsf{val}^*_3, 0, 256), \mathsf{val}^*_2 \| \mathsf{val}^*_3, i, j),$$

with G_{tuak} and G^*_{tuak} from $\{0,1\}^\kappa \times \{0,1\}^d \times \{0,1\}^t \times \{0,1\}^t$ to $\{0,1\}^n$, $\mathsf{val} = (\mathsf{val}_1 \| \mathsf{val}_2) \| \mathsf{val}_3 \in \{0,1\}^{512} \times \{0,1\}^{256} \times \{0,1\}^{(832-\kappa)}$, $\mathsf{val}^* = (\mathsf{val}^*_1 \| \mathsf{val}^*_2) \| \mathsf{val}^*_3 \in \{0,1\}^{256} \times \{0,1\}^{256} \times \{0,1\}^{(1088-\kappa)}$ two known values with $n = j - i$, $d = 1600 - \kappa$, $\kappa = |K|$ and $\log_2(t-1) < 1600 \leq \log_2(t)$, K a secret value and $0 \leq i \leq j \leq 1600$. The updated TUAK algorithms are generalized as the same way including the value $\mathsf{cst}_5 = \mathsf{Id_S} \| \mathsf{Pad} \| 1 \| 0^{192}$, We express the required security properties of the generalization G_{tuak} (resp. G^*_{tuak}) under the prf-security of the function f (resp. f^*). Since the construction of the two functions, while we cannot prove the latter property, we can conjecture that the advantage of a prf-adversary would be of the form:

$$\mathsf{Adv}^{\mathsf{prf}}_{f^*}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{prf}}_{f}(\mathcal{A}) \leq \mathsf{c}_1 \cdot \frac{t/T_f}{2^{|K|}} + \mathsf{c}_2 \cdot \frac{q \cdot t/T_f}{2^{1600-m}},$$

for any adversary $\mathcal{A}$ running in time t and making at most q queries at its challenger. Here, m is the output's size of our function f and T_f is the time to do one f computation on the fixed RAM model of computation and c_1 and c_2 are two constants depending only on this model. In other words, we assume that the best attacks are either a exhaustive key search or a specific attack on this construction. This attack uses the fact that the permutation is public and can be easily inverted. Even if the protocol truncates the permutation, if the output values are large, and an exhaustive search on the missing bits is performed, it is possible to invert the permutation and recover the inputs. Since the secret keys is one of the inputs as well as some known values are also inputs, it is then possible to determine which guesses of the exhaustive search are correct guess or incorrect ones. Finally, if the known inputs are shorter than the truncation, false positives can happen due to collisions and we have to filter the bad guesses. However, if the number of queries is large enough, it is possible to filter these bad guesses and uniquely recover the keys.

Pseudorandomness of TUAK algorithms. We begin by reducing the prf-security of G_{tuak} to the prf-security of the function f. This implies the prf-security of each TUAK algorithm. Recall that our main assumption is that the function f is prf-secure if the Keccak permutation is a good random permutation.

Theorem 15. *[prf-security for G^*_{tuak}.] Let $G^*_{\mathsf{tuak}} : \{0,1\}^\kappa \times \{0,1\}^e \times \{0,1\}^{d-e} \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^n$ and $f^* : \{0,1\}^\kappa \times \{0,1\}^e \times \{0,1\}^d \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^m$ be the two functions specified above. Consider a (t,q)-adversary $\mathcal{A}$ against the prf-security of the function G^*_{tuak}, running in time t and making at most q queries to its challenger. Denote the advantage of this adversary as $\mathsf{Adv}^{\mathsf{prf}}_{G^*_{\mathsf{tuak}}}(\mathcal{A})$. Then there exists a $(t' \approx O(t), q' = q)$-adversary $\mathcal{A}'$ with an advantage $\mathsf{Adv}^{\mathsf{prf}}_{f^*}(\mathcal{A}')$*

of winning against the pseudorandomness of f^* such that:

$$\mathsf{Adv}^{\mathsf{prf}}_{G^*_{\mathsf{tuak}}}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{prf}}_{f^*}(\mathcal{A}'),$$

Theorem 16. *[prf-**security for** G_{tuak}.] Let $G_{\mathsf{tuak}} : \{0,1\}^{\kappa} \times \{0,1\}^{e} \times \{0,1\}^{d-e} \times \{0,1\}^{t} \times \{0,1\}^{t} \rightarrow \{0,1\}^{n}$ and $f : \{0,1\}^{\kappa} \times \{0,1\}^{e} \times \{0,1\}^{d} \times \{0,1\}^{t} \times \{0,1\}^{t} \rightarrow \{0,1\}^{m}$ be the two functions specified above. Consider a (t,q)-adversary $\mathcal{A}$ against the prf-security of the function G_{tuak}, running in time t and making at most q queries to its challenger. Denote the advantage of this adversary as $\mathsf{Adv}^{\mathsf{prf}}_{G_{\mathsf{tuak}}}(\mathcal{A})$. Then there exist a $(t' \approx 2{\cdot}t, q' = 2{\cdot}q)$-adversary $\mathcal{A}$' with an advantage $\mathsf{Adv}^{\mathsf{prf}}_{f}(\mathcal{A}')$ of winning against the pseudorandomness of f such that:*

$$\mathsf{Adv}^{\mathsf{prf}}_{G_{\mathsf{tuak}}}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{prf}}_{f}(\mathcal{A}').$$

The security of (classic and updated) MILENAGE.

In order to prove the prf-security of the MILENAGE algorithms, we assume that the AES permutation is a good pseudo-random function.

We model the AES algorithm by the function f and a keyed version of a classic Davies-Meyer by the function f^*:

$$f(K,x) = \mathsf{AES}_K(x), f^*(K,x) = K \oplus \mathsf{AES}_x(K),$$

with $x \in \{0,1\}^{128}$, $K \in \{0,1\}^{\kappa}$. Contrary to the TUAK algorithms, the MILENAGE algorithms have not the same behavior. Let the construction G_{mil1} (resp. G^*_{mil1}), the generalization of the functions $\mathcal{F}_1$ and $\mathcal{F}^*_1$ and G_{mil2} (resp. G^*_{mil2}) the generalization of the functions $\mathcal{F}_2$, $\mathcal{F}_3$, $\mathcal{F}_4$, $\mathcal{F}_5$, $\mathcal{F}^*_5$ which are keyed with the subscriber key $\mathsf{sk_C}$ (resp. with the operator key $\mathsf{sk_{Op}}$):

$$\begin{aligned}
\mathcal{F}_1(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Sqn}, \mathsf{AMF}) &= G_{\mathsf{mil1}}(\mathsf{sk_C}, \mathsf{inp}_1, 0, 63) = G^*_{\mathsf{mil1}}(\mathsf{sk_{Op}}, \mathsf{inp}^*_1, 0, 63),\\
\mathcal{F}^*_1(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Sqn}, \mathsf{AMF}) &= G_{\mathsf{mil1}}(\mathsf{sk_C}, \mathsf{inp}_2, 64, 127) = G^*_{\mathsf{mil1}}(\mathsf{sk_{Op}}, \mathsf{inp}^*_2, 64, 127),\\
\mathcal{F}_2(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R}) &= G_{\mathsf{mil2}}(\mathsf{sk_C}, \mathsf{inp}_2, 64, 127) = G^*_{\mathsf{mil2}}(\mathsf{sk_{Op}}, \mathsf{inp}^*_2, 64, 127),\\
\mathcal{F}_3(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R}) &= G_{\mathsf{mil2}}(\mathsf{sk_C}, \mathsf{inp}_3, 0, 127) = G^*_{\mathsf{mil2}}(\mathsf{sk_{Op}}, \mathsf{inp}^*_3, 0, 127),\\
\mathcal{F}_4(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R}) &= G_{\mathsf{mil2}}(\mathsf{sk_C}, \mathsf{inp}_4, 0, 127) = G^*_{\mathsf{mil2}}(\mathsf{sk_{Op}}, \mathsf{inp}^*_4, 0, 127),\\
\mathcal{F}_5(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R}) &= G_{\mathsf{mil2}}(\mathsf{sk_C}, \mathsf{inp}_2, 0, 47) = G^*_{\mathsf{mil2}}(\mathsf{sk_{Op}}, \mathsf{inp}^*_2, 0, 47),\\
\mathcal{F}^*_5(\mathsf{sk_{Op}}, \mathsf{sk_C}, \mathsf{R}) &= G_{\mathsf{mil2}}(\mathsf{sk_C}, \mathsf{inp}_5, 0, 47) = G^*_{\mathsf{mil2}}(\mathsf{sk_{Op}}, \mathsf{inp}^*_5, 0, 47),
\end{aligned}$$

with:

$\mathsf{inp}_1 = \mathsf{sk_{Op}} \| \mathsf{R} \| (\mathsf{Sqn} \| \mathsf{AMF}) \| \mathsf{c}_1 \| \mathsf{r}_1 \| 0^{128}, \mathsf{inp}^*_1 = \mathsf{sk_C} \| \mathsf{R} \| (\mathsf{Sqn} \| \mathsf{AMF}) \| \mathsf{c}_1 \| \mathsf{r}_1 \| 0^{128}$,
$\forall i \in \{2, ..., 5\}, \mathsf{inp}_i = \mathsf{sk_{Op}} \| \mathsf{R} \| \mathsf{c}_i \| \mathsf{r}_i \| 0^{128}, \mathsf{inp}^*_i = \mathsf{sk_C} \| \mathsf{R} \| \mathsf{c}_i \| \mathsf{r}_i \| 0^{128}$.

For the updated MILENAGE algorithms, we use the same constructions G_{mil1} and G_{mil2} as follows:

$$\mathbf{Upd}\_\mathsf{F}_1(\mathsf{sk_{Op}},\mathsf{sk_C},\mathsf{R},\mathsf{Sqn},\mathsf{AMF}) = G_{\mathsf{mil1}}(\mathsf{sk_C},\mathsf{inp}_1,0,127) = G^*_{\mathsf{mil1}}(\mathsf{sk_{Op}},\mathsf{inp}^*_1,0,127),$$
$$\mathbf{Upd}\_\mathsf{F}^*_1(\mathsf{sk_{Op}},\mathsf{sk_C},\mathsf{R},\mathsf{Sqn},\mathsf{AMF}) = G_{\mathsf{mil1}}(\mathsf{sk_C},\mathsf{inp}_6,0,127) = G^*_{\mathsf{mil1}}(\mathsf{sk_{Op}},\mathsf{inp}^*_6,0,127),$$
$$\mathbf{Upd}\_\mathsf{F}_2(\mathsf{sk_{Op}},\mathsf{sk_C},\mathsf{R}) = G_{\mathsf{mil2}}(\mathsf{sk_C},\mathsf{inp}_2,0,127) = G^*_{\mathsf{mil2}}(\mathsf{sk_{Op}},\mathsf{inp}^*_2,0,127),$$
$$\mathbf{Upd}\_\mathsf{F}_3(\mathsf{sk_{Op}},\mathsf{sk_C},\mathsf{R}) = G_{\mathsf{mil2}}(\mathsf{sk_C},\mathsf{inp}_3,0,127) = G^*_{\mathsf{mil2}}(\mathsf{sk_{Op}},\mathsf{inp}^*_3,0,127),$$
$$\mathbf{Upd}\_\mathsf{F}_4(\mathsf{sk_{Op}},\mathsf{sk_C},\mathsf{R}) = G_{\mathsf{mil2}}(\mathsf{sk_C},\mathsf{inp}_4,0,127) = G^*_{\mathsf{mil2}}(\mathsf{sk_{Op}},\mathsf{inp}^*_4,0,127),$$
$$\mathbf{Upd}\_\mathsf{F}_5(\mathsf{sk_{Op}},\mathsf{sk_C},\mathsf{R}) = G_{\mathsf{mil2}}(\mathsf{sk_C},\mathsf{inp}_5,0,47) = G^*_{\mathsf{mil2}}(\mathsf{sk_{Op}},\mathsf{inp}^*_5,0,47),$$
$$\mathbf{Upd}\_\mathsf{F}^*_5(\mathsf{sk_{Op}},\mathsf{sk_C},\mathsf{R}) = G_{\mathsf{mil2}}(\mathsf{sk_C},\mathsf{inp}_5,80,47) = G^*_{\mathsf{mil2}}(\mathsf{sk_{Op}},\mathsf{inp}^*_5,80,47),$$

with:

$\mathsf{inp}_1 = \mathsf{sk_{Op}}\|\mathsf{R}\|(\mathsf{Sqn}\|\mathsf{AMF})\|\mathsf{c}_1\|\mathsf{r}_1\|\mathsf{Id_S}, \mathsf{inp}^*_1 = \mathsf{sk_C}\|\mathsf{R}\|(\mathsf{Sqn}\|\mathsf{AMF})\|\mathsf{c}_1\|\mathsf{r}_1\|\mathsf{Id_S},$
$\forall i \in \{2,...,6\}, \mathsf{inp}_i = \mathsf{sk_{Op}}\|\mathsf{R}\|\mathsf{c}_i\|\mathsf{r}_i\|\mathsf{Id_S}, \mathsf{inp}^*_i = \mathsf{sk_C}\|\mathsf{R}\|\mathsf{c}_i\|\mathsf{r}_i\|\mathsf{Id_S}.$

Then, these both constructions are constructed as follows:

$$G_{\mathsf{mil1}}(K,\mathsf{val}^{(1)},a,b) = \lfloor \mathsf{Top}_C \oplus f(K,\mathsf{val}_4 \oplus f(K,\mathsf{Top}_C \oplus \mathsf{val}_2 \oplus \mathsf{val}_6)\oplus \mathsf{Rot}_{\mathsf{val}_5}(\mathsf{Top}_C \oplus (\mathsf{val}_3\|\mathsf{val}_3)))\rfloor_{a..b},$$
$$G_{\mathsf{mil2}}(K,\mathsf{val}^{(2)},a,b) = \lfloor \mathsf{Top}_C \oplus f(K,\mathsf{val}_4\oplus \mathsf{Rot}_{\mathsf{val}_5}(\mathsf{Top}_C \oplus f(K,\mathsf{Top}_C \oplus \mathsf{val}_2 \oplus \mathsf{val}_6)))\rfloor_{a..b},$$
$$G^*_{\mathsf{mil1}}(K^*,\mathsf{val}^{*(1)},a,b) = \lfloor \mathsf{Top}_C \oplus f(\mathsf{val}^*_1,\mathsf{val}^*_4 \oplus f(\mathsf{val}^*_1,\mathsf{Top}_C \oplus \mathsf{val}^*_2 \oplus \mathsf{val}^*_6)\oplus \mathsf{Rot}_{\mathsf{val}^*_5}(\mathsf{Top}_C \oplus (\mathsf{val}^*_3\|\mathsf{val}^*_3)))\rfloor_{a..b},$$
$$G^*_{\mathsf{mil2}}(K^*,\mathsf{val}^{*(2)},a,b) = \lfloor \mathsf{Top}_C \oplus f(\mathsf{val}^*_1,\mathsf{val}^*_4\oplus \mathsf{Rot}_{\mathsf{val}^*_5}(\mathsf{Top}_C \oplus f(\mathsf{val}^*_1,\mathsf{Top}_C \oplus \mathsf{val}^*_2 \oplus \mathsf{val}^*_6)))\rfloor_{a..b},$$

with G_{mil1} (resp. G^*_{mil1}): $\{0,1\}^\kappa \times \{0,1\}^{d_1} \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^n$, G_{mil2} (resp. G^*_{mil2}): $\{0,1\}^\kappa \times \{0,1\}^{d_2} \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^n$, $\mathsf{val}^{(1)} = \mathsf{val}_1\|\ \mathsf{val}_2\|\ \mathsf{val}_3\|\ \mathsf{val}_4\|\ \mathsf{val}_5\|\|\mathsf{val}_6$, $\mathsf{val}^{(2)} = \mathsf{val}_1\|\ \mathsf{val}_2\|\ \mathsf{val}_4\|\ \mathsf{val}_5\|\mathsf{val}_6$, val_1,val_2, $\mathsf{val}_4, \mathsf{val}_6 \in \{0,1\}^{128}$,$\mathsf{val}_3 \in \{0,1\}^{64}$, $\mathsf{val}_5 \in \{0,1\}^7$, and $\mathsf{val}^{*(1)} = \mathsf{val}^*_1\|\ \mathsf{val}^*_2\|\ \mathsf{val}^*_3\|\ \mathsf{val}^*_4\|\ \mathsf{val}^*_5 \|\mathsf{val}^*_6$, $\mathsf{val}^{(2)}_* = \mathsf{val}^*_1\|\ \mathsf{val}^*_2\|\ \mathsf{val}^*_4\|\ \mathsf{val}^*_5\ \|\mathsf{val}^*_6$, val^*_1,val^*_2, $\mathsf{val}^*_4, \mathsf{val}^*_6 \in \{0,1\}^{128}$,$\mathsf{val}^*_3 \in \{0,1\}^{64}$, $\mathsf{val}^*_5 \in \{0,1\}^7$ and $\mathsf{Top}_C = \mathsf{val}_1 \oplus f(K,\mathsf{val}_1) = K^* \oplus f^*(\mathsf{val}^*_1,K^*)$.

We express the security property of the generalizations G_{mil1} and G_{mil2} (resp. G^*_{mil1} and G^*_{mil2}) under the prf-security of the function f (resp. f^*). While we cannot prove the latter property, we can conjecture that the advantage of a prf-adversary would be of the form:

$$\mathsf{Adv}^{\mathsf{prf}}_f(\mathcal{A}) \leq \mathsf{c}_1 \cdot \frac{t/T_f}{2^{128}} + \mathsf{c}_2 \cdot \frac{q^2}{2^{128}},$$

for any adversary $\mathcal{A}$ running in time t and making at most q queries at its challenger. Here, m is the output's size of our function f and T_f is the time to do one f computation on the fixed RAM model of computation and c_1 and c_2 are two constants depending only on this model. In other words, we assume that the best attacks are either a exhaustive key search or a linear cryptanalysis. We also conjecture that the advantage of a prf-adversary on f^* is negligible.

Pseudorandomness of MILENAGE algorithms. We begin by reducing the prf-security of G_{mil1} and G_{mil2} to the prf-security of the function f. This implies the prf-security of each MILENAGE algorithm.

Theorem 17. *[prf-security for G_{mil1} and G_{mil2}]*
Let G_{mil1} (resp. G_{mil2}): $\{0,1\}^{\kappa} \times \{0,1\}^{d} \times \{0,1\}^{t} \times \{0,1\}^{t} \rightarrow \{0,1\}^{n}$ and f : $\{0,1\}^{\kappa} \times \{0,1\}^{128} \rightarrow \{0,1\}^{128}$ be the two functions specified above. Consider a (t,q)-adversary $\mathcal{A}$ against the prf-security of the function G_{mil1} (resp. G_{mil2}), running in time t and making at most q queries to its challenger. Denote the advantage of this adversary as $\mathsf{Adv}^{\mathsf{prf}}_{G_{\mathsf{mil1}}}(\mathcal{A})$. Then there exists a $(t' \approx 3 \cdot t, q' = 3 \cdot q)$-adversary $\mathcal{A}$' with an advantage $\mathsf{Adv}^{\mathsf{prf}}_{f}(\mathcal{A}')$ of winning against the pseudorandomness of f such that:

$$\mathsf{Adv}^{\mathsf{prf}}_{G_{\mathsf{mil1}}}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{prf}}_{f}(\mathcal{A}') (= \mathsf{Adv}^{\mathsf{prf}}_{G_{\mathsf{mil2}}}(\mathcal{A})).$$

Theorem 18. *[prf-security for G^{*}_{mil1} and G^{*}_{mil2}]*
Let G^{*}_{mil1} (resp G^{*}_{mil2}): $\{0,1\}^{\kappa} \times \{0,1\}^{d} \times \{0,1\}^{t} \times \{0,1\}^{t} \rightarrow \{0,1\}^{n}$ and f^{*} : $\{0,1\}^{\kappa} \times \{0,1\}^{128} \rightarrow \{0,1\}^{128}$ be the two functions specified above. Consider a (t,q)-adversary $\mathcal{A}$ against the prf-security of the function G^{*}_{mil1} (resp. G^{*}_{mil2}), running in time t and making at most q queries to its challenger. Denote the advantage of this adversary as $\mathsf{Adv}^{\mathsf{prf}}_{G^{*}_{\mathsf{mil1}}}(\mathcal{A})$ (resp. $\mathsf{Adv}^{\mathsf{prf}}_{G^{*}_{\mathsf{mil2}}}(\mathcal{A})$). Then there exists a $(t' \approx O(t), q' = q)$-adversary $\mathcal{A}$' with an advantage $\mathsf{Adv}^{\mathsf{prf}}_{f^{*}}(\mathcal{A}')$ of winning against the pseudorandomness of f^{*} such that:

$$\mathsf{Adv}^{\mathsf{prf}}_{G^{*}_{\mathsf{mil1}}}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{prf}}_{f^{*}}(\mathcal{A}') (= \mathsf{Adv}^{\mathsf{prf}}_{G^{*}_{\mathsf{mil2}}}(\mathcal{A})).$$

F.1 Formal Analysis of the AKA Protocol

ProVerif is an automatic protocol verifier in the Dolev-Yao formal model; it handles protocols represented by Horn clauses and can prove that they have various security properties (including key secrecy, indistinguability, and (mutual) authentication). ProVerif can analyze a wide range of asymmetric and symmetric stateless protocols, and can run an unbounded number of sessions of an AKE protocol with an unbounded message space, thanks to some well-chosen approximations.

As a consequence, while it can also give some false negatives (attacks which do not always translate to our model), if the verifier outputs a security proof, then the latter carries over automatically. When such a proof cannot be generated, the tool tries to propose an execution trace, i.e. a specific attack, which "breaks" the desired property. Cryptographic primitives are modeled as generic functions and reduction rules. Indeed, for example, the symmetric encryption algorithms are modeled by `func senc \ 2` and the reduction `reduc sdec(k, senc(k, m))`, where `senc` and `sdec` are respectively symmetric encryption and decryption, models the property that the plaintext m can be retrieved from the ciphertext and the private key k.

The syntax of the ProVerif calculus processes is given by the following grammar:

P, Q, R ::=	plain processes
0	null process
P|Q	parallel composition
!P	replication
new n; P	name restriction
if M = N then P else Q	conditional
in(M, x); P	message input
out(M, N); P	message output.

The null process signifies inaction, while P|Q represents the parallel execution of P and Q. The replication !P of a process P behaves as the parallel execution of an unbounded number of copies of P. The restriction new n; P creates a new name n whose scope is restricted to the process P, and it then runs P. The message input in(M, x); P (respectively the message output out(M, x); P)) describes a process that receives (respectively sends) a message x from the channel M and after behaves as P. The conditional checks the equality between N and M and behaves as P if the answer is true; otherwise Q is executed. See [8] for more details. We used the ProVerif tool to analyze the mutual authentication and the key derivation security of the real AKA protocol. This analysis is clearly different to the stateless feature of the variant studied by Arapinis and al. [7]. Indeed, the latter swaps the sequence number of AKA protocol for a random value. Usually, the stateful feature of AKA protocol is incompatible with ProVerif[8].

Despite its restriction to stateless protocols, we nonetheless try to adapt the behavior of the ProVerif calculus to obtain some results on AKA. ProVerif does not keep any trace between several sessions. Thus, we have implemented (by duplication) the code of each session in each process for as many as sessions as there are, hardcoding the sequence number. Since we cannot duplicate an unbounded number of sessions, we have to restrict the verifier to bound this number. This leaves us unable to apply the replication of the process. While both security notions (authentication and key derivation) are modelled by different injective correspondence properties, they are always proved under a replication. So, ProVerif cannot give any efficient result about AKA protocol.

As an alternative, we used a verifier of stateful processes, called StatVerif. The latter is an extension of the ProVerif process calculus constructed for explicit state; its goal is to bridge the gap, and handle stateful protocols. We report the most relevant parts of the StatVerif scripts used for the verification of the AKA protocol. We omit the declaration of constants, functions, and restriction rules, and report only the code of the both process.

Unfortunately, for unknown reasons, the sequence is incremented *ad infinitum*. So, StatVerif does not give some exploitable results.

[8] This could explain the stateless consideration of [7].

Mobile Client Process:

```
let processA=
out(c, UID);
lock(state);
read state as xState;
let next = incr(xState) in
unlock(state);
in(c, (rand, m2, m3));
let f3a = prf1(Kab, rand) in
let f5a = prf3(Kab, rand) in
let xxstate = sdec(m2, f5a) in
if xxstate = next then
let (= rand, = next) = checkmac1(m3, Kab) in
let Ks = f3a in
out(c, mac2(rand, Kab));
0.
```

Global Process:

```
process
(!(processA)|!(processB))
```

Server process:

```
let processB=
lock(state);
read state as xState;
let next = incr(xState) in
unlock(state);
in(c, = UID);
new rb;
let f3 = prf1(Kab, rb) in
let f5 = prf3(Kab, rb) in
let chiff = senc(next, f5) in
let mac = mac1((rb, next), Kab) in
out(c, (rb, chiff, mac));
in(c, res);
if res = mac2(rb, Kab) then
let Ks = f3in
lock(state);
state := next;
unlock(state);
0.
```

Fig. 12. AKA Procedure in StatVerif: