A Cryptographic Analysis of UMTS/LTE AKA

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

Transmitting confidential and authenticated data between two parties across an insecure channel is a fundamental goal in cryptography. Secure channels are usually obtained by means of an authenticated key-exchange (AKE) protocol.

AKE protocols generally consist of two phases. During the first phase, the parties authenticate each other and exchange data that enables them to compute a master key. The latter is then used to derive one or several secret keys, as well as

other useful values. In a second phase, the derived keys are used to construct the secure channel between the parties, guaranteeing the confidentiality, integrity, and authentication of the data they exchange.

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[1]. The AKA protocol is used in a greater context in the 3rd Generation Partnership Project (3GPP), which aims to develop the specifications for the next generation mobile systems. The security of the system is covered by Technical Specifications 33 (TS 33) and 35 (TS 35)[2], from both an architectural and a security-algorithm standpoint.

The AKA protocol. Initially developed in the 1990s, AKA uses symmetric keys exclusively, in a mobile-network context which imposes a peculiar architecture. In this setup, mobile *clients* subscribe to a single *operator*, which provides them with mobile services (messaging, calls, Internet use, etc.). Services are provided across a secure channel, not by the operator, but by an intermediate *local* network operator (which we call *server* to avoid confusion). The server and operator are affiliated together for domestic use. However, if the client is abroad, service is provided by a server affiliated with a different operator. Thus, servers are only trusted to provide services, but they must not learn the client's long-term secrets (known only to the client and the operator); by contrast, servers do learn short-term secret values, such as session keys, which are necessary for the transmission of the required service. Consequently, unlike the classical two-party AKE setting, the AKA protocol requires three participants.

One specificity of the subscriber-operator architecture is that clients are associated both with a unique client-key and with their operator's key, which is shared with all the other clients (a potentially very large number) of that operator. Clients minimize the risk of compromising the shared key by only storing a (one-way) function of that, and the client key, never the operator key in clear.

The design of the AKA protocol is influenced by three important constraints. One is that (current and older) SIM cards, cannot generate (pseudo)random numbers. Thus, freshness has to be guaranteed without client randomness. The second constraint is that the (necessary) communication[3] between servers and operators in the roaming scenario is usually expensive. In the AKA protocol, operators generate *batches* of *authentication vectors* for the server, thus minimizing costs. Finally, mobile channels are notoriously noisy, requiring the protocol to be robust with respect to noise. As a result of these constraints, the AKA protocol is *stateful*, with the authentication depending on an updatable *sequence number*, which is accepted within a tolerance interval.

[1] We stress that while AKA is an instance of authenticated key-exchange, AKE denotes a larger class of protocols, including e.g. TLS/SSL, PACE/EAC, etc.

[2] See `http://www.3gpp.org/DynaReport/33-series.htm` and `http://www.3gpp.org/DynaReport/35-series.htm`.

[3] Notably, since the server is not trusted, it needs information from the client's operator to provide service to the client.

TUAK **and** MILENAGE**.** In this paper we focus on the *provable security* of AKA. The latter is constructed using symmetric-key primitives, namely a set of seven cryptographic functions, denoted $\mathcal{F}_1, \ldots, \mathcal{F}_5, \mathcal{F}_1^*, \mathcal{F}_5^*$. We closely follow the design of these algorithms, as well as that of the protocol, in our analysis.

Originally, 3GPP put forward a proposal for an AES-encryption-based algorithm set, called MILENAGE [1]. As an alternative to MILENAGE, the ETSI SAGE committee proposed another set of algorithms called TUAK [2], which relies on a truncation of Keccak's internal permutation. The winner of the SHA-3 hash function competition, Keccak offers both higher performance, in hardware and software, than AES, and resistance to many generic attacks. While the TUAK algorithms inherit Keccak's superior performance, they do not use the Keccak permutation in a usual, black-box way, but rather rely on something akin to a Merkle-Damgård construction. Instead, the internal permutation is truncated, then used in a cascade, which makes previous results harder to use. We cannot simply use the same assumptions for the truncated version as we would for the original permutation, either. Our analysis of the key indistinguishability, as well as client- and respectively server-impersonation resistance of the protocol concerns both the classical MILENAGE-based version, and the one using TUAK.

Related Work. At its core, the AKA protocol provides authenticated key exchange (AKE), a primitive first modelled by Bellare and Rogaway [14]. We use the Bellare-Pointcheval-Rogaway (BPR) extension of this framework [13] in our definitions; however, the three-party setting and lack of randomness on the prover side do not allow us to simply "import" their model, as we explain in more detail below.

Few papers give a security proof for AKA, especially when instantiated with MILENAGE. Gilbert provides an out-of-context security proof for MILENAGE [12], showing it operates as a kind of counter mode in deriving key materials. It is unclear whether this suffices to guarantee security for AKA at large; indeed, as we show in this paper, the AES-based design is not quite as versatile as TUAK. The closest results to a security proof of AKA (see below) use automated (formal) verification.

In 2003, Zhang [16] described an important server-corruption attack against AKA and advised against the use of sequence numbers as state. He also presented a stateless modification of the protocol called AP-AKA Server corruptions are a highly relevant threat in a post-Snowden cryptographic era, in which intelligence agencies have been known to substitute and backdoor algorithms, and store massive amounts of data. We take such attacks into account, and define security in a game-based framework akin to the BPR model, allowing for easier proofs and more nuances in the definitions. We also extend a countermeasure from Zhang [16], which features the addition, in the authentication string, of a unique server-specific identifier, and we show how to incorporate it within the existent MILENAGE and TUAK specifications.

The cryptographic analysis of Lee et al. [15] is complementary to ours as they focus in detail on the LTE (Long-term Evolution) protocol, used in 4G networks (similar to AKA, but using different identifiers and key-management),

rather than the handshake itself. Lee et al. analyse the *privacy* of LTE, rather than the *security* of AKA (as is the case in our paper). Their main result is that in the absence of server corruptions, LTE attains a weak degree of untraceability against an active MiM adversary. The notion is modelled in terms of real/ideal world indistinguishability, in which the adversary must trace a client using multiple temporary identifiers (denoted TMSI as opposed to IMSI, which denotes permanent user identifiers) across different sessions. Though this is not made explicit, Lee et al.'s result implies the impersonation resistance of LTE (else, the adversary can force desynchronizations); some guarantee is also implicit for the security of the derived session keys (else, this affects the TMSI reallocation). However, their results hold for an important modification of AKA. A surprising problem is that [15] cannot capture IMSI-catcher attacks (which directly impact privacy *without* server corruptions); this is because they assume that once a TMSI is allocated, the IMSI will never again appear in clear. However, in the real AKA scheme, an active attacker can replace a sent TMSI with a random message and thus *force* the server to request the client's IMSI. Lee et al. also do not model re-synchronizations, which are problematic in privacy analyses. The two-party model they consider (client–server, rather than client–server–operator) is justified in the absence of corruptions, but incomplete with respect to real-world circumstances. Whereas they consider client-specific secrets, there is no mention of the shared operator key. Finally, their proofs reduce the privacy of AKA to some assumptions on the functions which are akin to the unitary function G that we use; however, they do not analyse TUAK and MILENAGE to verify whether these suites actually guarantee those required properties.

Arapinis et al. [6] focus on the client privacy of the AKA protocol by means of automated verification using ProVerif [7]; however, they only assess a *modified* version, which randomizes the sequence number. This fundamental modification makes their results inapplicable to the original protocol. Our attempts to extend this analysis to that of the true protocol by using StatVerif [18][4] were not fruitful, as we discuss in Appendix F.1.

Our contributions. We present four main contributions: (a) fully-formalized definitions for the security of AKA in the three party setting; (b) security proofs indicating that the current AKA protocol does not attain full security in the presence of server corruptions (due to the attack of Zhang [16]); (c) we show how to attain full security by simply adding a unique server identifier in the authentication; (d) we prove that our security statements hold for both protocol instantiations (TUAK and MILENAGE). In particular, we analyse two somewhat-similar versions of the protocol: the original AKA scheme and a slight variation of it of our own design, which we also analyse. We detail our contributions below.
Security Model. We first define a threat model and five game-based security notions for the 3-party AKA protocol, two with respect to a Man-in-the-Middle adversary (akin to BPR security, but with three parties, and allowing the option

[4] This is an extension of ProVerif proposed by Arapinis et al. to handle automatic verification for protocols with global state.

of server corruptions in the Strong, as opposed to the Weak property), and two with respect to malicious servers. These properties are:

1. **key-indistinguishability:** i.e. the derived session keys are indistinguishable from random by a Man-in-the-Middle (MiM) attacker acting between the client and a server, and which has black-box access to all operators.
2. **client- and server-impersonation:** i.e. a MiM attacker cannot impersonate honest servers (to the client), or clients (to an honest server). Due to the identification phase, the AKA protocol resists client-impersonation better than server-impersonation.
3. **state-confidentiality:** i.e. (malicious) servers do not learn the client's secret key, nor state, nor the operator's secret key. We assume that the server interacts with both operators and clients, but we only address the AKA handshake (not the secure-channel primitives).
4. **soundness:** i.e. (malicious) servers cannot make the client accept the server's authentication (thus completing the key derivation process), unless they are explicitly given authenticating information by a legitimate operator.

Security Proofs. We analyse the security of two versions of AKA: the current one, and our modification of it. In the full version, we also show that the AP-AKA version of the protocol, due to Zhang, is vulnerable to a replay attack. We prove that, under the assumption that the seven cryptographic functions behave as a unitary function G that is pseudorandom when keyed with the client key, the *current* AKA version guarantees: weak key-indistinguishability; weak server-impersonation resistance; strong client-impersonation resistance; and soundness. If furthermore the algorithms behave as a PRF called G^*, when keyed with the operator key, AKA also guarantees state confidentiality. For our modification of the AKA protocol, we prove, under the same assumptions: state-indistinguishability, soundness, as well as strong key-indistinguishability, server- and client-impersonation security. This first proof step, reducing protocol security to that of a unitary function, allows us to define a sufficient security requirement for the underlying sub-algorithms.

TUAK and MILENAGE. The second step of the proof is to show that both TUAK and MILENAGE behave as the required functions G and G^*. This can be proved for TUAK under the standard assumption that the (un-)truncated Keccak permutation is a good PRF [8,10]. By contrast, proving that MILENAGE can be modelled as a unitary PRF when keyed with the operator key requires the pseudorandomness of a keyed AES-version of a classic Davies-Meyer construction for MILENAGE, which seems a stronger assumption than just assuming the pseudorandomness of the underlying AES permutation.

AKA privacy. Several papers indicate privacy problems for AKA, e.g. [9,19,6,5]. The last of these is a recent result, indicating that privacy can be attacked at a lower level than the protocol layer (by leakage at a physical layer). Since AKA is known not to provide strong privacy, and it is moreover unclear whether it *can* even hope to provide it considering such leaks at lower layers, we choose to restrict ourselves to the subject of AKA security, rather than its privacy.

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

Mobile 3G networks use the variant of AKA fully depicted in Figure 6, allowing the client and the server to output session keys $(\mathsf{CK}, \mathsf{IK})$, which are then used to secure future message-exchanges. The same protocol is the backbone of the 4G LTE protocol; however, for LTE the client is associated with an identifier called GUTI (see 3GPP TS 23.003, release 13), as opposed to the tuple of permanent and temporary identifiers we describe below. The use of GUTIs make no difference for our analysis. More significantly, the session keys CK, IK from the 3G protocol are only used as key material for a key derivation function KDF, which outputs the true session key.[5] Our proofs work similarly for this new key derivation, but we would need an additional reduction to KDF security.

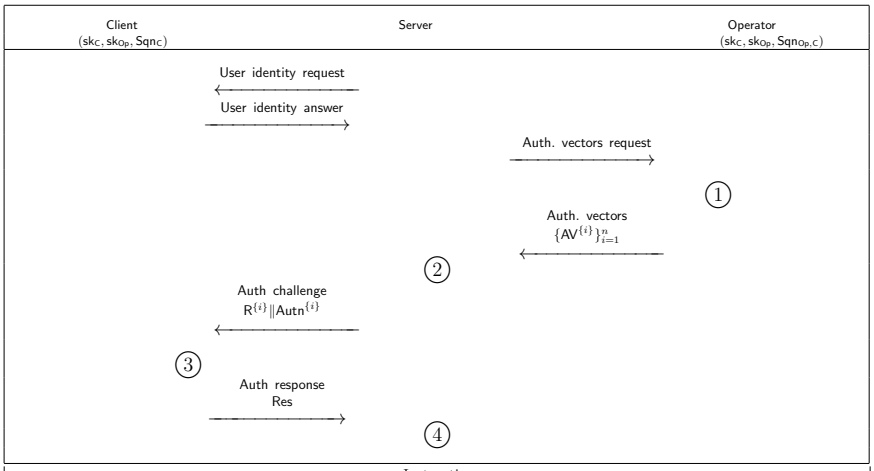
This protocol features two main *active* actors: the client (in 3GPP terminology $\mathsf{ME/USIM}$) and the server (denoted VLR). The third, only selectively-active party is the operator (denoted HLR). The tripartite setup of AKA was meant for roaming, for which the server providing mobile coverage 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 ideal; however, the server/operator communication is (financially) expensive.

Section 3 describes in detail the setup of the three parties. Clients C and operators Op use both the client's secret key $\mathsf{sk_C}$ and the operator's secret key $\mathsf{sk_{Op}}$[6]. The client and operator also keep track of sequence numbers $\mathsf{Sqn_C}$ (resp. $\mathsf{Sqn_{Op,C}}$), updated after each successful authentication (by a simple, predictable procedure, e.g. incrementing them by a fixed value). If the states are

[5] This key, denoted $\mathsf{K_{asme}}$, is computed as: $\mathsf{K_{asme}} = \mathsf{KDF}(\mathsf{CK} \| \mathsf{IK}, \mathsf{ID_{SN}}, \mathsf{Sqn} \oplus \mathsf{AK}, \mathsf{const})$, with $\mathsf{ID_{SN}}$ the serving operator network identity.

[6] 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.

too far apart, the client prompts a re-synchronization. The three parties: clients, servers, and operators, also know the client's permanent identifier IMSI. Clients and servers must keep track of tuples (IMSI, TMSI, LAI), the last two values forming a unique temporary identifier, which is updated at every session.



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.

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 the tuple of a Temporary Mobile Subscriber Identity TMSI and the Local Area Identifier LAI of the server issuing the latest TMSI. The identification procedure is vital to the client's privacy; however, as we focus here only on the *security* of AKA, we just associate each client with a unique user ID UID (as we explain at the end of this section). Once the server associates 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 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 features the algorithms $\mathcal{F}_1^*, \mathcal{F}_5^*$ for re-synchronization. All algorithms take as input the client key $\mathsf{sk_C}$, the operator 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, described below. Else, 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 a TMSI *re-allocation*. During the optional *re-synchronization*, the client uses $\mathsf{Sqn_C}$ to compute values $\mathsf{Mac_S^*}$ and $\mathsf{AK}^* \oplus \mathsf{Sqn_C}$ as Op did, using the session R, but algorithms $\mathcal{F}_1^*$ and $\mathcal{F}_5^*$ (not $\mathcal{F}_1$ 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.

Following successful key exchange, the client and server run the TMSI *re-allocation*. The server sends an (unauthenticated) encryption of a new, randomly chosen TMSI (which is unique per server) to the client C, using the agreed-upon key CK. Encryption is done by means of the A5/3 algorithm (see 3GPP TS 43.020, release 12), 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 $\mathsf{TMSI_{new}}$ and $\mathsf{TMSI_{old}}$ are retained and can be used in the next protocol run.

Identities and reallocation. Though in this paper we stick close to the AKA protocol, one simplification we make throughout is associating each client with a single, unique UID, which we consider public. In practice, UID is the user's IMSI, which is used 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 with IMSI values.

Another important feature of AKA 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 we currently do. As discussed 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 impersonate servers by offline relays. The use of the TMSI in AKA is undone by using IMSIs as a backup for TMSIs; also, insecurities in using TMSIs translate to the identification by IMSI.

3 Security model

3.1 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. [14,13]. 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 $\mathsf{st_{UID}}$[7]. In particular, we consider multiple operators, with the restriction that each user may only be registered to a single operator[8]. In our model, we also assume for simplicity that the key space of all operators is identical, noting that neither the

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

[8] 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.

key-indistinguishability, nor the mutual authentication properties are affected 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 *adversarially 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)[9]. 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

[9] In this we keep faithful to the implementation of AKA, which protects $\mathsf{sk}_{\mathsf{Op}}$ from the user by storing a 1-way function of $\mathsf{sk}_{\mathsf{Op}}$ and sk_{C} in the SIM card. Another approach would be to reveal an intermediate, AKA-specific value denoted Top_C upon corruption. In the interest of generality, we keep the model at a higher level of abstraction than the peculiarities of AKA. We also note that in our proofs, a common first step is to give the adversary access to a broader corruption oracle, which also reveals $\mathsf{sk}_{\mathsf{Op}}$, with no security loss.

returning the message that the operator Op would have output to the server on input a client C.
- $\mathsf{StReveal}(\mathsf{C}, i, \mathsf{bit}_\mathsf{S}) \rightarrow x$: 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 [10]. 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:*

[10] The choice of pid and sid makes our security guarantee non-composable; however, the design of AKA makes it hard to define pids based only on publicly-known values.

(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[11]. 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.

[11] 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.

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* $\mathsf{Adv}_{\Pi}^{\mathsf{K.Ind}}(\mathcal{A}) > \epsilon$.

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. [20], 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 $\mathsf{Execute}$ 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:*** $\mathsf{C.Imp}$ ***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.

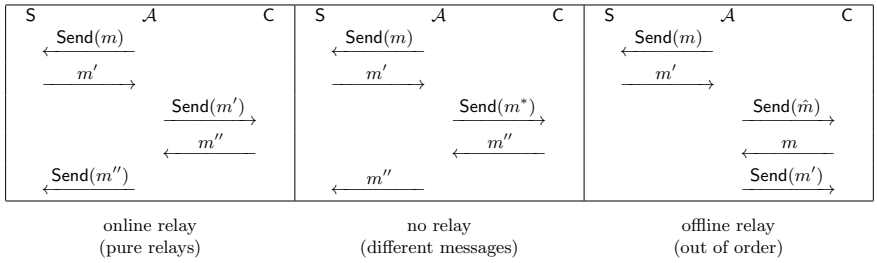


Fig. 2. Examples of Online and Offline relays. For the AKA protocol, the message m is the client's UID, which the adversary can learn. The message m' is a valid authentication challenge, and the message m'' is the authentication response. The message $\hat{m}$ is the UID request message, whereas m^* is a random message.

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

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

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* $\mathsf{Adv}_{\Pi}^{\mathsf{C.Imp}}(\mathcal{A}) \geq \epsilon$*.*

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* $\mathsf{Send}(m)$ *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 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} OpAccess *queries per operator per corrupted server for strong security), and making at most* q_{G} *queries to the function* G*, has an advantage* $\mathsf{Adv}_{\Pi}^{\mathsf{S.Imp}}(\mathcal{A}) \geq \epsilon$*.*

3.2 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: state-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**, Execute, Send, Reveal, StReveal oracles as described in the previous model. We additionally add two new oracles (including a new 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 $S = (\mathsf{UID}, \mathsf{sk}_{\mathsf{UID}}, \mathsf{st}_{\mathsf{UID}}, \mathsf{st}_{\mathsf{Op},\mathsf{C}})$ 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 state 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: $(\mathsf{P}_i, \mathsf{sk}^*_{\mathsf{UID}}, \mathsf{sk}^*_{\mathsf{Op}}, \mathsf{st}^*_{\mathsf{UID}}, \mathsf{st}^*_{\mathsf{Op},\mathsf{UID}})$ 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: $\mathsf{sk}^*_{\mathsf{UID}}, \mathsf{sk}^*_{\mathsf{Op}}, \mathsf{st}^*_{\mathsf{UID}}, \mathsf{st}^*_{\mathsf{Op},\mathsf{UID}}$ is respectively equal to $\mathsf{sk}_{\mathsf{UID}}, \mathsf{sk}_{\mathsf{Op}}, \mathsf{st}_{\mathsf{UID}}, \mathsf{st}_{\mathsf{Op},\mathsf{UID}}$, 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 it to the 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 must also 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_{C}. 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 TUAK and MILENAGE, guarantee this property.

We note that the pseudorandomness of G implies the pseudorandomness of each sub-algorithm, but is a strictly stronger property, which is necessary since 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. As we detail in the full paper, the exact values of d, t, and n differ for TUAK and MILENAGE; however, the construction of G is somewhat similar.

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.

Security statements. We group the five security statements that we prove for the AKA protocol into two theorems. The first groups the properties of: weak key-indistinguishability, strong client- and weak server-impersonation resistance, and soundness with respect to servers. The second theorem is that for state-confidentiality, which requires an additional assumption. We only include proof sketches in this section and the complete proofs are moved in Appendix D. Indeed, we sketch below the proof of each of these properties. For every proof, we will require several reductions, which we formulate in game hops, starting from an original game $\mathbb{G}_0$ which is the original security game for that property, as defined in Section 3. Our security statements are phrased with respect to an adversary $\mathcal{A}$ trying to break (in some way) the security of Π, which runs in time t, creates at most q_{exec} party instances with at most q_{res} resynchronizations per instance, and makes at most q_G queries to the function G. Furthermore, in the case of strong MiM adversary, it can also corrupt at most q_{s} servers and make at most q_{Op} OpAccess queries per operator per corrupted server. For the *legitimate-and-malicious* adversary, we quantify $\mathcal{A}$ in terms of the maximal number q_{Op} of queries to the oracle OpAccess, and the similar q_{exec}, q_{res} and q_G queries.

The function G is defined as above.

Theorem 1. [W.K.Ind**-resistance**] *For the protocol Π using the unitary function G described above, for any $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_G)$-adversary $\mathcal{A}$ against the* W.K.Ind*-security of Π winning with advantage $\mathsf{Adv}_{\Pi}^{\mathsf{W.K.Ind}}(\mathcal{A})$ there exists a $(t' \approx O(t), q' =$*

$q_G + q_{\mathsf{exec}}(2 + q_{\mathsf{res}})$)*-adversary* $\mathcal{A}'$ *against the pseudorandomness of* G *with:*

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

Proof. We use the following game hops:

$\mathbb{G}_0$**:** The adversary in this game is an active MiM, whose goal is to distinguish the session keys of a fresh party instance P_i ending in an accepted state from random keys of the same length. No server corruptions are allowed.

$\mathbb{G}_0 \Rightarrow \mathbb{G}_1$**:** We modify the original game $\mathbb{G}_0$ as follows: we extend the corruption oracle such that the adversary can request to corrupt the operator key $\mathsf{sk}_{\mathsf{Op}}$. 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).

$\mathbb{G}_1 \Rightarrow \mathbb{G}_2$**:** In $\mathbb{G}_2$, $\mathcal{A}$ can only interact with a single client. This is equivalent to guessing the challenge client and our security loss is a factor $\frac{1}{n_{\mathsf{C}}}$.

$\mathbb{G}_2 \Rightarrow \mathbb{G}_3$**:** In $\mathbb{G}_3$, we abort if two honest server instances output the same random value R. The two games are equivalent up to a collision term $\frac{q^2_{\mathsf{exec}}}{2^{|\mathsf{R}|}}$.

$\mathbb{G}_3 \Rightarrow \mathbb{G}_4$**:** In $\mathbb{G}_4$, we replace the output of G by truly random, but consistent values. The security loss here is precisely the advantage of a distinguisher $\mathcal{A}'$ for the pseudorandomness of G.

$\mathbb{G}_4$**:** In this game, $\mathcal{A}$ plays against a single client C, using unique R values. Its goal is to distinguish random session keys from truly random, real, fresh keys. The adversary can thus do no better than guess.

Theorem 2. [S.C.Imp**-resistance**] *For the protocol* Π *using the unitary function* G *described above, for any* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{s}}, q_{\mathsf{Op}}, q_G)$*-adversary* $\mathcal{A}$ *against the* S.C.Imp*-security of* Π*, winning with advantage* $\mathsf{Adv}^{\mathsf{S.C.Imp}}_{\Pi}(\mathcal{A})$*, there exists a* $(t' \approx O(t), q' = 5 \cdot q_{\mathsf{s}} \cdot q_{\mathsf{Op}} + q_G + q_{\mathsf{exec}}(q_{\mathsf{res}} + 2))$*-adversary* $\mathcal{A}'$ *against the pseudorandomness of* G *such that:*

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

Proof. We use the following game hops:

$\mathbb{G}_0$**:** In this game, the adversary is again an active MiM, whose goal is to make an honest server instance S_i end in an accepting state with partner id UID and session sid if we exclude online relays for sid. Up to q_{s} server corruptions are allowed.

$\mathbb{G}_0 \Rightarrow \mathbb{G}_3$**:** We use the same game hops from the previous proof to reach to $\mathbb{G}_3$, in which: $\mathcal{A}$ can interact with a single client and there are no R collisions. However, as we can now corrupt servers, the security loss from $\mathbb{G}_2$ to $\mathbb{G}_3$ is a term $\frac{(q_{\mathsf{exec}} + q_{\mathsf{s}} \cdot q_{\mathsf{Op}})^2}{2^{|\mathsf{R}|}}$.

$\mathbb{G}_3 \Rightarrow \mathbb{G}_4$**:** We modify $\mathbb{G}_3$ by only allowing the adversary to interact with a single server (which cannot be corrupted). In particular, the adversary cannot reuse authentication vectors from corrupted servers to provide a response to the uncorrupted server S. However, the server's challenge in any new session is associated with a fresh, unique random value R, which enters into each G computation, including the authentication challenge and response. Thus, our security loss is a term $\mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}')$ (collision of G output for 2 different inputs implies distinguishing the output of G from random).

$\mathbb{G}_4 \Rightarrow \mathbb{G}_5$**:** We modify $\mathbb{G}_4$ by replacing the pseudorandom output of G by consistent, but truly random output and we lose the advantage of the distinguisher $\mathcal{A}'$ against the pseudorandomness of G.

$\mathbb{G}_5$**:** At this point, the adversary plays the impersonation game against a single client and server, with only truly random, but consistent values, for unique R values. The adversary is left with three options: (1) re-using a value already received from the honest client; (2) guessing the key $\mathsf{sk_C}$; (3) guessing the response Res in each of the at most $q_{\mathsf{exec}} \cdot q_{\mathsf{res}}$ sessions with S. Option (1) requires online relays and is thus ruled out. Option (2) succeeds with probability $2^{-|\mathsf{sk_C}|}$. Option (3) succeeds w.p. $2^{-|\mathsf{Res}|}$ per guessing attempt, yielding a total term $q_{\mathsf{exec}} \cdot q_{\mathsf{res}} \cdot 2^{-|\mathsf{Res}|}$.

Theorem 3. [W.S.Imp**-resistance**] *For the protocol* Π *using the unitary function* G *described above, for any* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{G}})$*-adversary* $\mathcal{A}$ *against the* W.S.Imp*-security of* Π*, winning with advantage* $\mathsf{Adv}^{\mathsf{W.S.Imp}}_{\Pi}(\mathcal{A})$*, there exists a* $(t' \approx t, q = q_{\mathsf{exec}} \cdot (q_{\mathsf{res}} + 2) + q_{\mathsf{G}})$*-adversary* $\mathcal{A}'$ *against the pseudorandomness of* 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 use the following game hops:

$\mathbb{G}_0$**:** In this game, the adversary is an active MiM, whose goal is to make an honest client instance C_i end in an accepting state with partner ID S and session ID sid such that there are no offline *or* online relays. No server corruptions are allowed.

$\mathbb{G}_0 \Rightarrow \mathbb{G}_2$**:** We use the same game hops from the previous proof to reach to $\mathbb{G}_2$, in which: $\mathcal{A}$ can interact with a single client.

$\mathbb{G}_2 \Rightarrow \mathbb{G}_3$**:** We modify $\mathbb{G}_2$ by replacing the pseudorandom output of G by consistent, but truly random output and we lose the advantage of the distinguisher $\mathcal{A}'$ against the pseudorandomness of G.

$\mathbb{G}_3$**:** At this point, the adversary plays the impersonation game against a single client and server, with only truly random, but consistent values. The adversary's goal is to generate a correct authentication challenge, in particular the value Autn for a correct sequence number. The adversary is left with three options: (1) re-using a value already received from the honest server; (2) guessing the key $\mathsf{sk_C}$; (3) guessing a consistent value Autn in each of the at most $q_{\mathsf{exec}} \cdot q_{\mathsf{res}}$ sessions with C. Option (1) requires an offline or online

relay and is thus excluded. Option (2) succeeds with probability $2^{-|\mathsf{sk_C}|}$. Option (3) succeeds w.p. $2^{-|\mathsf{Mac_S}|}$ per guessing attempt, yielding a total term $q_{\mathsf{exec}} \cdot q_{\mathsf{res}} \cdot 2^{-|\mathsf{Mac_S}|}$.

Theorem 4. [Sound**-resistance**] *For the protocol* Π *using the unitary function* G *described above, for any* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{Op}}, q_{\mathsf{G}}, \epsilon)$*-adversary* $\mathcal{A}$ *against the soundness of* Π*, winning with advantage* $\mathsf{Adv}^{\mathsf{Sound}}_{\Pi}(\mathcal{A})$*, there exists a* $(t' \approx t, q' = 5 \cdot q_{\mathsf{Op}} + q_G + n_{\mathsf{C}} \cdot q_{\mathsf{exec}}(2 + q_{\mathsf{res}}))$*-adversary* $\mathcal{A}'$ *against the pseudorandomness of* G *such that:*

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

Proof. We use the following game hops:

$\mathbb{G}_0$**:** This game is similar to that of S.S.Imp, except that S.S.Imp is defined with respect to an active MiM and soundness, with respect to a malicious server. This server receives q_{Op} authentication vectors and must authenticate to the client q_{Op} times. Apart from the original malicious server, no other server is corrupted.

$\mathbb{G}_0 \Rightarrow \mathbb{G}_1$**:** We define game $\mathbb{G}_1$ as the modification of the S.S.Imp game in which $\mathcal{A}$ also has access to operator corruptions and can get up to q_{Op} authentication vectors; the adversary's goal is to authenticate $q_{\mathsf{Op}} + 1$ times. We show that for every adversary winning game $\mathbb{G}_0$ with some probability, there exists an adversary winning game $\mathbb{G}_1$ with the same probability. Intuitively this is why the $q_{\mathsf{Op}} + 1$-st authentication challenge cannot be produced by the adversary except by impersonating the server as a MiM would.

$\mathbb{G}_1$**:** We look at the probability that $\mathcal{A}$ wins game $\mathbb{G}_1$, in particular with respect to the $q_{\mathsf{Op}} + 1$-st session, for which it has no authentication challenges. The adversary's attack is then reduced to a W.S.Imp model, for which it can ask no additional authentication vector. This yields a total success probability equal to the W.S.Imp bound.

Theorem 5. [St.Conf**-resistance**] *For the protocol* Π *using the unitary functions* G, G^**, for any* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{Op}}, q_G, q_{G^*})$*-adversary* $\mathcal{A}$ *against the* St.Conf*-security of* Π*, winning with advantage* $\mathsf{Adv}^{\mathsf{St.Conf}}_{\Pi}(\mathcal{A})$*, 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^* *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. We use several game hops for this proof, as follows:

$\mathbb{G}_0$**:** As in the soundness game, the adversary is a malicious server, whose goal is to predict at least one of four values for any client UID: $\mathsf{sk_{UID}}, \mathsf{sk_{Op}}, \mathsf{Sqn_{UID}}$, $\mathsf{Sqn_{Op,UID}}$, the latter two values taken at the time of the adversary's output.

$\mathbb{G}_0 \Rightarrow \mathbb{G}_1$**:** We modify the original game $\mathbb{G}_0$ so as to include only a single operator. We construct a black-box reduction that yields no security loss at this game hop. Intuitively, this is because we consider G as a PRF keyed with the secret key $\mathsf{sk}_{\mathsf{UID}}$.

$\mathbb{G}_1 \Rightarrow \mathbb{G}_2$**:** In $\mathbb{G}_2$, $\mathcal{A}$ can only interact with a single client. In this game hop, an interesting difficulty is simulating queries for clients other than the target client, which are affiliated to the same operator. We can do this by requiring that we model the AKA algorithms behave as a PRF when keyed with the operator key (this is the PRF property of the function we call G^*). Thus, the adversary requires two oracles of the functions G^* and G.

$\mathbb{G}_2 \Rightarrow \mathbb{G}_3$**:** In $\mathbb{G}_3$, we replace the output of G and G^* by truly random, but consistent values. The security loss here is precisely the advantage of a distinguisher $\mathcal{A}'$ for the pseudorandomness of G or G^*.

$\mathbb{G}_4$**:** In this game, $\mathcal{A}$ plays the game against a single client C, using truly random values. Her goal is to output at least one correct long-term secret for the remaining client UID. Since all exchanged values are truly random, the adversary has only a guessing probability of at most $\frac{1}{2^{|\mathsf{sk_C}|}} + \frac{1}{2^{|\mathsf{sk_{Op}}|}} + \frac{2}{2^{|\mathsf{Sqn}|}}$.

MILENAGE **and** TUAK**.** Our second step is to prove that TUAK and MILENAGE can both be represented as the generic function G. Due to space constraints, we only propose two theorems of the pseudorandomness of these functions and leave all the details to the proofs in Appendix F. Notably, as opposed to TUAK (whose symmetric design allows a lot more leeway), the MILENAGE algorithms require a stronger assumption to prove the PRF property for G^* (which is keyed with $\mathsf{sk}_{\mathsf{Op}}$).

Theorem 6. [prf**-security for** TUAK **algorithms**] *For the generalization of the* TUAK *algorithms* G_{tuak} *(resp.* G^*_{tuak}*) keyed with the subscriber key (resp. the operator key) and the functions* f *and* f^* *two different truncated keyed internal permutation of* Keccak*, for any* (t, q)*-adversary* $\mathcal{A}$ *against the pseudorandomness of the function* f *(resp.* f^**), then there exists a* $(t' \approx t, q' = q)$*-adversary* $\mathcal{A}$*' such that:*

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

Theorem 7. [prf**-security for** MILENAGE **algorithms**] *For the generalization of the* MILENAGE *algorithms* G_{mil1} *and* G_{mil2} *(resp.* G^*_{mil1} *and* G^*_{mil2}*) keyed with the subscriber key (resp. the operator key) and the function* f *(resp.* f^**) the* AES *algorithm (resp. a keyed version of a classic Davies-Meyer), for any* (t, q)*-adversary* $\mathcal{A}$ *against the pseudorandomness of the function* f *(resp.* f^**), then there exists a* $(t' \approx 3 \cdot t, q' = 3 \cdot q)$*-adversary* $\mathcal{A}$*' such that:*

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

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 [16] and Zhang and Fang [17] 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.

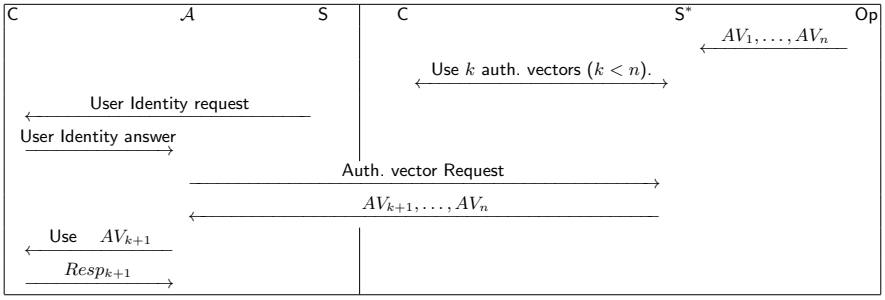


Fig. 3. The attack of Zhang and Fang. On the right hand side, the client is in the vulnerable network, interacting with the server S^*. The server uses up authentication vectors $\mathsf{AV}_1, \ldots \mathsf{AK}_k$. Then, the server S^* is corrupted, and the adversary $\mathcal{A}$ learns $\mathsf{AV}_{k+1},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

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.

Such attack do 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 challenges 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 present our modified protocol in Figure 4.

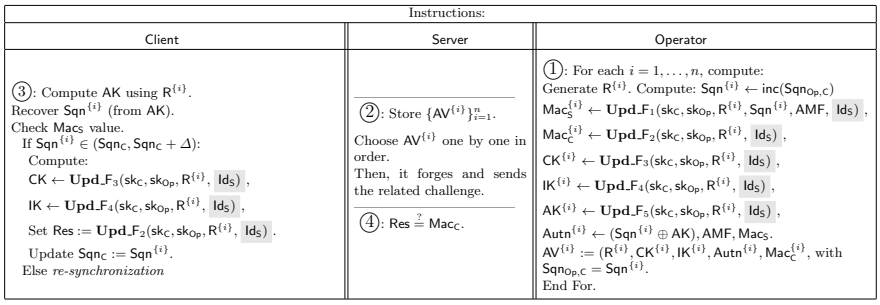


Fig. 4. The modified instructions of our variant.

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. As the previous theorems, we only include proof sketches and the complete proofs are moved in appendix D.

Theorem 8. [S.K.Ind**-resistance**] *For the modified protocol* Π *using the unitary function* G *described in Section 4, for any* $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{s}}, q_{\mathsf{Op}}, q_G)$*-adversary* $\mathcal{A}$ *against the* S.K.Ind*-security of* Π *winning with advantage* $\mathsf{Adv}_{\Pi}^{\mathsf{S.K.Ind}}(\mathcal{A})$ *there exists a* $(t' \approx O(t), q' = 5 \cdot q_{\mathsf{s}} \cdot q_{\mathsf{Op}} + q_G + q_{\mathsf{exec}}(q_{\mathsf{res}} + 2))$*-adversary* $\mathcal{A}'$ *against the*

pseudorandomness of G with:

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

Proof. We use the following game hops:

$\mathbb{G}_0$**:** The adversary in this game is an active MiM, whose goal is to distinguish the session keys of a fresh party instance P_i ending in an accepted state from random keys of the same length. We allow server corruptions.

$\mathbb{G}_0 \Rightarrow \mathbb{G}_2$**:** We use the same reductions as in the weak key-indistinguishability game, until we only interact with one client.

$\mathbb{G}_2 \Rightarrow \mathbb{G}_3$**:** In $\mathbb{G}_3$, we abort if two honest server instances, or a server-corruption/ operator access output the same random value R. The two games are equivalent up to a collision term $\frac{(q_{\mathsf{exec}}+q_{\mathsf{s}}\cdot q_{\mathsf{Op}})^2}{2^{|\mathsf{R}|}}$.

$\mathbb{G}_3 \Rightarrow \mathbb{G}_4$**:** In $\mathbb{G}_4$, we modify the game to only play against one server. The security loss here comes from the adversary's no longer being able to corrupt servers. However, corruptions only give authentication vectors with a different server identifier. Thus, we bound the security loss by the advantage of a best distinguisher against the pseudorandomness of G.

$\mathbb{G}_4 \Rightarrow \mathbb{G}_5$**:** In this game hop, we replace the pseudorandom outputs of G by truly random values, thus losing the advantage of a best distinguisher against the PRF-security of G.

$\mathbb{G}_5$**:** In this game, $\mathcal{A}$ plays against a single client C and a single server S, using unique R values. Its goal is to distinguish random session keys from truly random, consistent, fresh keys. The adversary can thus do no better than guess.

Theorem 9. [S.S.Imp**-resistance**] *For the modified protocol Π using the unitary function G described in Section 4, for any $(t, q_{\mathsf{exec}}, q_{\mathsf{res}}, q_{\mathsf{s}}, q_{\mathsf{Op}}, q_{\mathsf{G}})$-adversary $\mathcal{A}$ against the* S.S.Imp*-security of Π, winning with advantage $\mathsf{Adv}^{\mathsf{S.S.Imp}}_{\Pi}(\mathcal{A})$, there exists a $(t' \approx O(t), q' = 5 \cdot q_{\mathsf{s}} \cdot q_{\mathsf{Op}} + q_G + q_{\mathsf{exec}}(2 + q_{\mathsf{res}}))$-adversary $\mathcal{A}'$ against the pseudorandomness of G such that:*

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

Proof. We use the following game hops:

$\mathbb{G}_0$**:** The adversary in this game is an active MiM, whose goal is to impersonate the server to a client instance C_i, with no online relays. We allow server corruptions.

$\mathbb{G}_0 \Rightarrow \mathbb{G}_2$**:** We use the same first reductions as in the strong key-indistinguishability game, until we only interact with one client.

$\mathbb{G}_2 \Rightarrow \mathbb{G}_4$**:** We use the reductions $\mathbb{G}_3 \Rightarrow \mathbb{G}_5$ from the strong key-indistinguishability game. Now $\mathcal{A}$ interacts with a single client, a single server, and the output of G is replaced by consistent, truly random output.

$\mathbb{G}_5$**:** By a similar result to the W.S.Imp game, we now conclude that the adversary may at most guess either $\mathsf{Mac_S}$ or $\mathsf{sk_C}$.

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 [16]. 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. Altaf Shaik and Ravishankar Borgaonkar and N. Asokan and Valtteri Niemi and Jean-Pierre Seifert. Practical attacks against privacy and availability in 4G/LTE mobile communication systems. In *Accepted to NDSS 2016*, 2016.
6. 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.
7. 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.
8. 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.
9. D.Strobel. IMSI Catcher. In *Seminar Work, Ruhr-Universitat Bochum*, 2007.
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 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.
12. Henri Gilbert. The Security of "One-Block-to-Many" Modes of Operation. In *Fast Software Encryption, 10th International Workshop, FSE*, pages 376–395, 2003.
13. 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.

14. 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.
15. Ming-Feng Lee and Nigel P. Smart and Bogdan Warinschi and Gaven J. Watson. Anonymity guarantees of the UMTS/LTE authentication and connection protocol. *Int. J. Inf. Sec.*, 13(6):513–527, 2014.
16. Muxiang Zhang. Provably-Secure Enhancement on 3GPP Authentication and Key Agreement Protocol. *IACR Cryptology ePrint Archive*, 2003:92, 2003.
17. 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.
18. 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.
19. Myrto Arapinis and Tom Chothia and Eike Ritter and Mark Ryan. Analysing Unlinkability and Anonymity Using the Applied Pi Calculus. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*, pages 107–121, 2010.
20. 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.

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.*

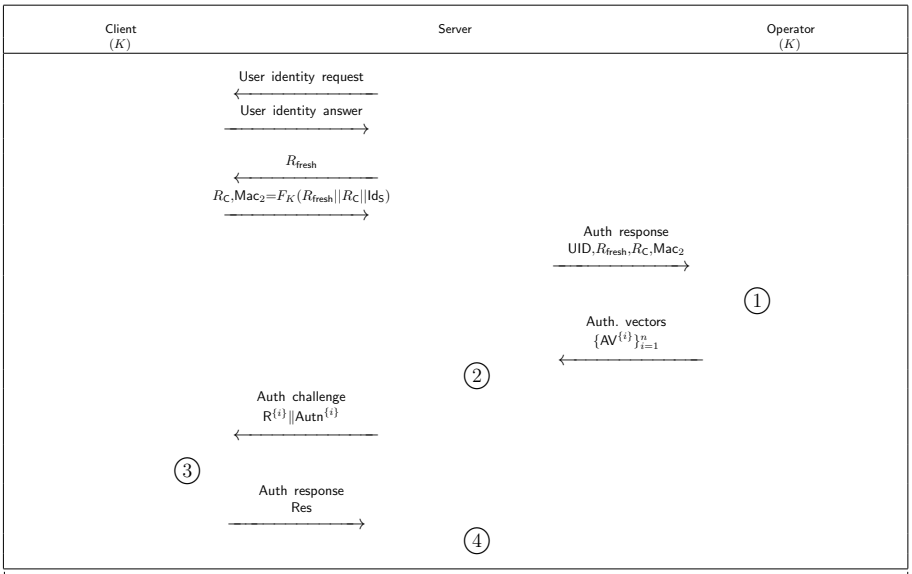
B The AP-AKA variant [16]

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.

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 [16] 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



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 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.

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 vectors 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 vectors from the operator and does so. Then, S^* may use several such vectors 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 [16], 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 [16] 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 internal cryptographic 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 these functions.

In 3G networks, 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). Both the ME/USIM and the HLR keep track of counters, denoted respectively Sqn_C and $\mathsf{Sqn}_{\mathsf{Op},\mathsf{C}}$; 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 sk_C and the same operator key 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 sk_C and the operator key 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 sk_C, the operator key sk_Op, and the session-specific random value R. Notice that the master key 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 deriving the operator key, the subscriber key 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.

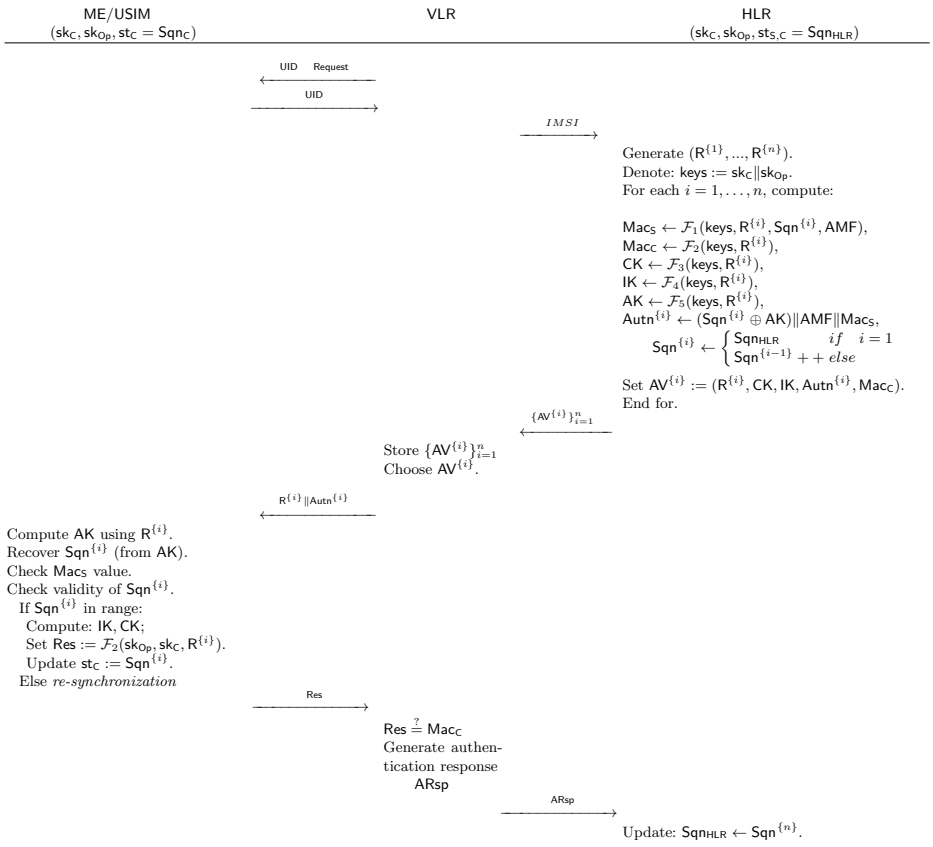


Fig. 6. The AKA Procedure.

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 challenge. 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 $\mathsf{Sqn_C}$[12]. 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 $\mathsf{ME/USIM}$. If the two previous verifications are successful i.e if the received authentication challenge is fresh, the network is authenticated by the $\mathsf{ME/USIM}$. Then, the $\mathsf{ME/USIM}$ 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 $\mathsf{ME/USIM}$. Otherwise, the VLR will initiate an *authentication failure* report procedure with the HLR. Note that the verification of the sequence number by the $\mathsf{ME/USIM}$ will cause the rejection of any attempt to re-use an authentication challenge 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 $\mathsf{ME/USIM}$ 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 synchronization failure message, the VLR does not immediately send a new user authentication request to the $\mathsf{ME/USIM}$, but rather notifies the HLR of the 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 $\mathsf{Sqn_{Op,C}}$ is in the correct range relatively to $\mathsf{Sqn_C}$. If the $\mathsf{Sqn_{Op,C}}$ verifies this property, it sends a new list of

[12] 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.

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

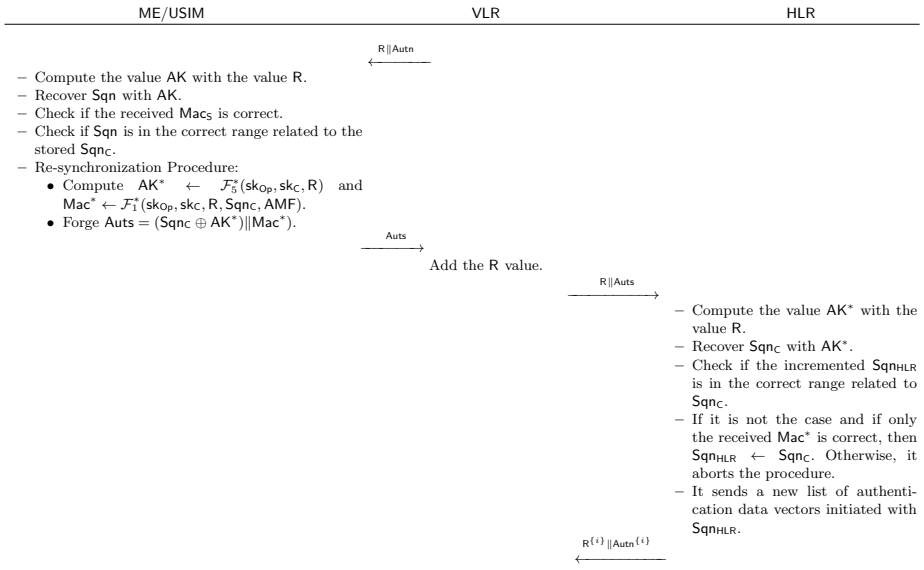


Fig. 7. The re-synchronization procedure of AKA protocol.

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: $\mathsf{sk}_{\mathsf{Op}}$ 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 $\mathsf{sk}_{\mathsf{Op}}$.

– 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

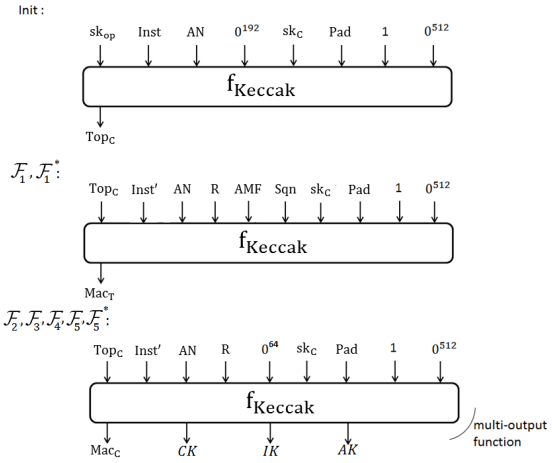


Fig. 8. TUAK diagram.

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 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 [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 $\mathsf{c_1}$,$\mathsf{c_2}$,$\mathsf{c_3}$,$\mathsf{c_4}$,$\mathsf{c_5}$ 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

Full proof of the theorem 1: W.K.Ind-resistance.

Our proof has the following hops.

Game $\mathbb{G}_0$: This game works as the 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:

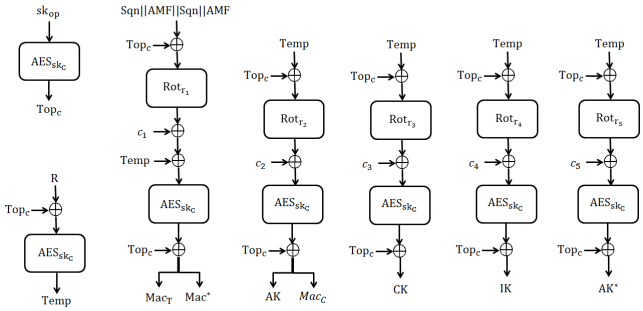


Fig. 9. MILENAGE diagram.

$\mathsf{Corrupt}(\mathsf{P}, \mathtt{type})$: 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

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

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).

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

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:

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

This concludes the proof. □

Full proof of the theorem 2: S.C.Imp-resistance.

Game $\mathbb{G}_0$: This game works as the 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 8. 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,

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

Security statement: This yields the following result:

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

□

Full proof of the theorem 3: W.S.Imp-resistance.

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:

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

□

Full proof of the theorem 4: Sound-resistance.

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 S.Imp-game but with a different adversary; indeed in the 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 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 $\mathsf{Adv}_{\Pi}^{\mathsf{Sound}}(\mathcal{A}_{\mathbb{G}_0})$, 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 $\mathsf{AV} = (\mathsf{R}, \mathsf{Autn}, \mathsf{Mac_C}, \mathsf{CK}, \mathsf{IK})$. 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 $\mathsf{Corrupt}$ 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 $\mathsf{sk_{Op}}$, and for each client of this operator the tuple $(\mathsf{UID}, \mathsf{sk_{UID}}, \mathsf{st_{Op,C}})$. 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 $\mathsf{C}$of 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 $\mathsf{S.S.Imp}$. Thus, we have :

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

□

Full proof of the theorem 5: $\mathsf{St.Conf}$-resistance**.**

Our proof has the following hops.

Game $\mathbb{G}_0$: This game works as the $\mathsf{St.Conf}$-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:

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

□

Full proof of the theorem 9: S.S.Imp-resistance.

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 $\mathsf{Corrupt}(\mathsf{P}, \mathsf{type})$, 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 8. 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

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

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:

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

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 $2^{-|\mathsf{Mac_S}|}$ per session (which is to say per instance and per re-synchronization), thus a total of $q_{\mathsf{exec}} \cdot q_{\mathsf{res}} \cdot 2^{-|\mathsf{Mac_S}|}$. Thus,

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

Security statement: This yields the following result:

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

Full proof of the theorem 8: S.K.Ind-resistance.

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. □

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 $\mathbf{Upd}\_\mathsf{F}_1$ and $\mathbf{Upd}\_\mathsf{F}_1^*$): a 48-bit sequence number.
- AMF (except for the functions $\mathbf{Upd}\_\mathsf{F}_1$ and $\mathbf{Upd}\_\mathsf{F}_1^*$): 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 **Upd_F**$_1$ and **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].

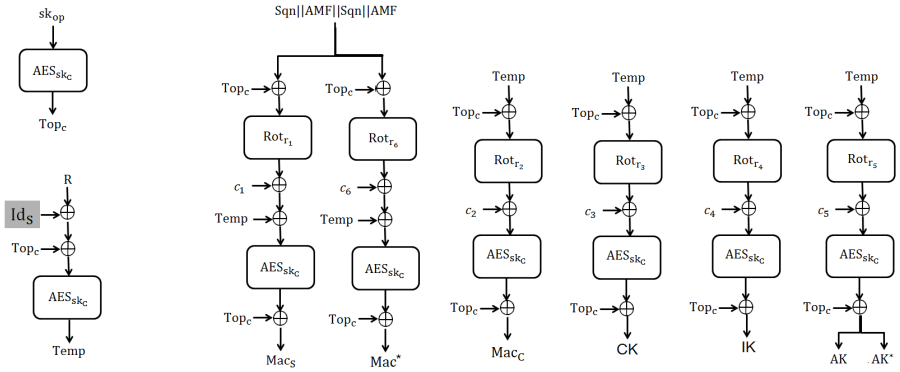


Fig. 10. Updated MILENAGE.

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:

$$\mathsf{Top}_C = \mathsf{sk_{Op}} \oplus \mathsf{AES_{sk_c}}(\mathsf{sk_{Op}}), \mathsf{Temp} = \mathsf{AES_{sk_c}}(\mathsf{R} \oplus \mathsf{Top}_C \oplus \mathsf{Id_S}).$$

The outputs of the MILENAGE algorithms are computed as follows:

- **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 [11]. 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 $\mathbf{Upd\_F}_1^*$ and $\mathbf{Upd\_F}_5^*$, 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 $\mathbf{Upd\_F}_1$ (resp. $\mathbf{Upd\_F}_1^*$) 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 $\mathsf{Id_S}$ and the smaller paging value Pad.

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

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

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

This is also depicted in Figure 11.

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

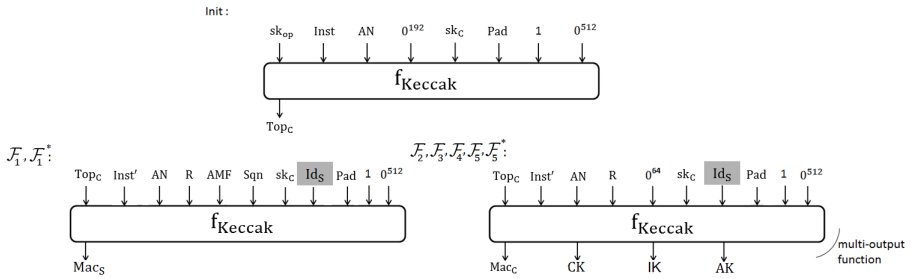


Fig. 11. Updated TUAK.

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:

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

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 $\mathsf{c_1}$ and $\mathsf{c_2}$ 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 10. *[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 11. *[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 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:
$\mathsf{inp}_1 = \mathsf{sk}_{\mathsf{Op}} \| \mathsf{R} \| (\mathsf{Sqn} \| \mathsf{AMF}) \| \mathsf{c}_1 \| \mathsf{r}_1 \| 0^{128}, \mathsf{inp}_1^* = \mathsf{sk}_{\mathsf{C}} \| \mathsf{R} \| (\mathsf{Sqn} \| \mathsf{AMF}) \| \mathsf{c}_1 \| \mathsf{r}_1 \| 0^{128}$,
$\forall i \in \{2, ..., 5\}, \mathsf{inp}_i = \mathsf{sk}_{\mathsf{Op}} \| \mathsf{R} \| \mathsf{c}_i \| \mathsf{r}_i \| 0^{128}, \mathsf{inp}_i^* = \mathsf{sk}_{\mathsf{C}} \| \mathsf{R} \| \mathsf{c}_i \| \mathsf{r}_i \| 0^{128}$.

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

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

with:
$\mathsf{inp}_1 = \mathsf{sk}_{\mathsf{Op}} \| \mathsf{R} \| (\mathsf{Sqn} \| \mathsf{AMF}) \| \mathsf{c}_1 \| \mathsf{r}_1 \| \mathsf{Id}_{\mathsf{S}}, \mathsf{inp}_1^* = \mathsf{sk}_{\mathsf{C}} \| \mathsf{R} \| (\mathsf{Sqn} \| \mathsf{AMF}) \| \mathsf{c}_1 \| \mathsf{r}_1 \| \mathsf{Id}_{\mathsf{S}}$,
$\forall i \in \{2, ..., 6\}, \mathsf{inp}_i = \mathsf{sk}_{\mathsf{Op}} \| \mathsf{R} \| \mathsf{c}_i \| \mathsf{r}_i \| \mathsf{Id}_{\mathsf{S}}, \mathsf{inp}_i^* = \mathsf{sk}_{\mathsf{C}} \| \mathsf{R} \| \mathsf{c}_i \| \mathsf{r}_i \| \mathsf{Id}_{\mathsf{S}}$.

Then, these both constructions are constructed as follows:

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

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}): $\{0,1\}^\kappa \times \{0,1\}^{d_2} \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^n$, $\mathsf{val}^{(1)} = \mathsf{val}_1 \| \mathsf{val}_2 \| \mathsf{val}_3 \| \mathsf{val}_4 \| \mathsf{val}_5 \| \mathsf{val}_6$, $\mathsf{val}^{(2)} = \mathsf{val}_1 \| \mathsf{val}_2 \| \mathsf{val}_4 \| \mathsf{val}_5 \| \mathsf{val}_6$, $\mathsf{val}_1, \mathsf{val}_2, \mathsf{val}_4, \mathsf{val}_6 \in \{0,1\}^{128}$, $\mathsf{val}_3 \in \{0,1\}^{64}$, $\mathsf{val}_5 \in \{0,1\}^7$, and $\mathsf{val}^{*(1)} = \mathsf{val}_1^* \| \mathsf{val}_2^* \| \mathsf{val}_3^* \| \mathsf{val}_4^* \| \mathsf{val}_5^* \| \mathsf{val}_6^*$, $\mathsf{val}_*^{(2)} = \mathsf{val}_1^* \| \mathsf{val}_2^* \| \mathsf{val}_4^* \| \mathsf{val}_5^* \| \mathsf{val}_6^*$, $\mathsf{val}_1^*, \mathsf{val}_2^*, \mathsf{val}_4^*, \mathsf{val}_6^* \in \{0,1\}^{128}$, $\mathsf{val}_3^* \in \{0,1\}^{64}$, $\mathsf{val}_5^* \in \{0,1\}^7$ 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:

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

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 $\mathsf{c_1}$ and $\mathsf{c_2}$ 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 12. *[prf-security for G_{mil1} and G_{mil2}]*
Let G_{mil1} (resp. G_{mil2}): $\{0,1\}^\kappa \times \{0,1\}^d \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^n$ and f : $\{0,1\}^\kappa \times \{0,1\}^{128} \to \{0,1\}^{128}$ 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 13. *[prf-security for G^*_{mil1} and G^*_{mil2}]*
Let G^*_{mil1} (resp G^*_{mil2}): $\{0,1\}^\kappa \times \{0,1\}^d \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^n$ and f^* : $\{0,1\}^\kappa \times \{0,1\}^{128} \to \{0,1\}^{128}$ 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 [7] 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. [6]. 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[13].

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

[13] This could explain the stateless consideration of [6].

protocol. We omit the declaration of constants, functions, and restriction rules, and report only the code of the both process.

Mobile Client Process:

```
1: let processA=
2: out(c, UID);
3: lock(state);
4: read state as xState;
5: let next = incr(xState) in
6: unlock(state);
7: in(c, (rand, m2, m3));
8: let f3a = prf1(Kab, rand) in
9: let f5a = prf3(Kab, rand) in
10: let xxstate = sdec(m2, f5a) in
11: if xxstate = next then
12: let (= rand, = next) = checkmac1(m3, Kab) in
13: let Ks = f3a in
14: out(c, mac2(rand, Kab));
15: 0.
```

Global Process:

```
1: process
2: (!(processA)|!(processB))
```

Server process:

```
1: let processB=
2: lock(state);
3: read state as xState;
4: let next = incr(xState) in
5: unlock(state);
6: in(c, = UID);
7:new rb;
8: let f3 = prf1(Kab, rb) in
9: let f5 = prf3(Kab, rb) in
10: let chiff = senc(next, f5) in
11: let mac = mac1((rb, next), Kab) in
12: out(c, (rb, chiff, mac));
13: in(c, res);
14: if res = mac2(rb, Kab) then
15: let Ks = f3in
16: lock(state);
17: state := next;
18: unlock(state);
19: 0.
```

Fig. 12. AKA Procedure in StatVerif:

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