

# A Cryptographic Analysis of UMTS/LTE AKA

Stephanie Alt<sup>1</sup>, Pierre-Alain Fouque<sup>2</sup>, Gilles Macariorat<sup>4</sup>,  
Cristina Onete<sup>3</sup>, and Benjamin Richard<sup>4</sup>(✉)

<sup>1</sup> DGA Bruz, Bruz, France  
s.alt@free.com

<sup>2</sup> IRISA, University of Rennes 1, Rennes, France  
pierre-alain.fouque@ens.fr

<sup>3</sup> INSA/IRISA Rennes, Rennes, France  
cristina.onete@gmail.com

<sup>4</sup> Orange Labs, Chatillon, France  
{gilles.macariorat,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 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<sup>1</sup>. The AKA protocol is used in a greater context in the 3<sup>rd</sup> 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)<sup>2</sup>, 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<sup>3</sup> 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

---

<sup>1</sup> 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.

<sup>2</sup> See <http://www.3gpp.org/DynaReport/33-series.htm> and <http://www.3gpp.org/DynaReport/35-series.htm>.

<sup>3</sup> Notably, since the server is not trusted, it needs information from the client's operator to provide service to the client.

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.

**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  $\mathcal{F}_1, \dots, \mathcal{F}_5, \mathcal{F}_1^*, \mathcal{F}_5^*$ . 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 [8]. We use the Bellare-Pointcheval-Rogaway (BPR) extension of this model [7]; 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 proof for MILENAGE [11], showing it operates as a kind of counter mode for key derivation. It is unclear whether this suffices to guarantee security for AKA at large; indeed, we show in this paper that MILENAGE 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 [15] 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 and proved its security in Shoup’s model. In the full version of this paper, we show that AP-AKA is still vulnerable to a particular type of replay attack. 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 into our definitions. We also extend a countermeasure from Zhang [15], 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 security proof of Lee et al. [13] is complementary to ours as they focus on the LTE (Long-term Evolution) protocol 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 we do). Their main result is that in the absence of server corruptions, LTE attains a weak untraceability against an active MiM adversary. We discuss their work in more detail in the full version. Though this is not made explicit, Lee et al.’s result implies the impersonation resistance of LTE and some security of the derived session keys; however, their proofs hold for an important modification of AKA, as we discuss in more detail in the full version. A surprising problem is that [13] cannot capture IMSI-catcher attacks (which directly impact privacy *without* server corruptions); this is because, contrary to real-world scenarios, they assume that once a TMSI is allocated, the IMSI will never again appear in clear. 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. [5] focus on the client privacy of the AKA protocol by automated verification in ProVerif [10]; 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 a state-permissive tool called StatVerif [6] were not fruitful, as discussed in the full paper.

**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 [15]); (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, three with respect to a Man-in-the-Middle –MiM– adversary (akin to BPR security, but with three parties, allowing server corruptions for the *strong*, as opposed to the *weak* property), and two with respect to malicious servers. These properties are:

1. **Key-indistinguishability:** the derived session keys are indistinguishable from random by a MiM attacker placed between the client and a server with black-box access to all operators.
2. **Client- and server-impersonation:** a MiM attacker cannot impersonate honest servers (to the client), or clients (to an honest server). Due to the identification phase, AKA resists client impersonations better than server impersonations.

3. **State-confidentiality:** (malicious) servers cannot learn: the client’s secret key, the operator’s secret key, nor their state. The malicious server may interact with both operators and clients, but we only address the AKA handshake (not the secure-channel primitives).
4. **Soundness:** (malicious) servers cannot authenticate to the client 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 [9, 12]. 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. [3–5, 14]. 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

Mobile 3G networks use the variant of AKA fully depicted in Fig. 1, allowing the client and the server to output session keys (CK, 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.<sup>4</sup> 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 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  $sk_C$  and the operator’s secret key  $sk_{Op}$ <sup>5</sup>. The client and operator also keep track of sequence numbers  $Sqn_C$  (resp.  $Sqn_{Op,C}$ ), updated after each successful authentication (by a simple, predictable procedure, e.g. incrementing them by a fixed value). If the states are 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.

The AKA protocol, depicted in Fig. 1, 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. 1), 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  $Mac_S$  (authenticating R and the value  $Sqn_{Op,C}$ ); a client-authentication string  $Mac_C$  (authenticating R only); the session keys CK

<sup>4</sup> This key, denoted  $K_{asme}$ , is computed as:  $K_{asme} = KDF(CK||IK, ID_{SN}, Sqn \oplus AK, const)$ , with  $ID_{SN}$  the serving operator network identity.

<sup>5</sup> 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.

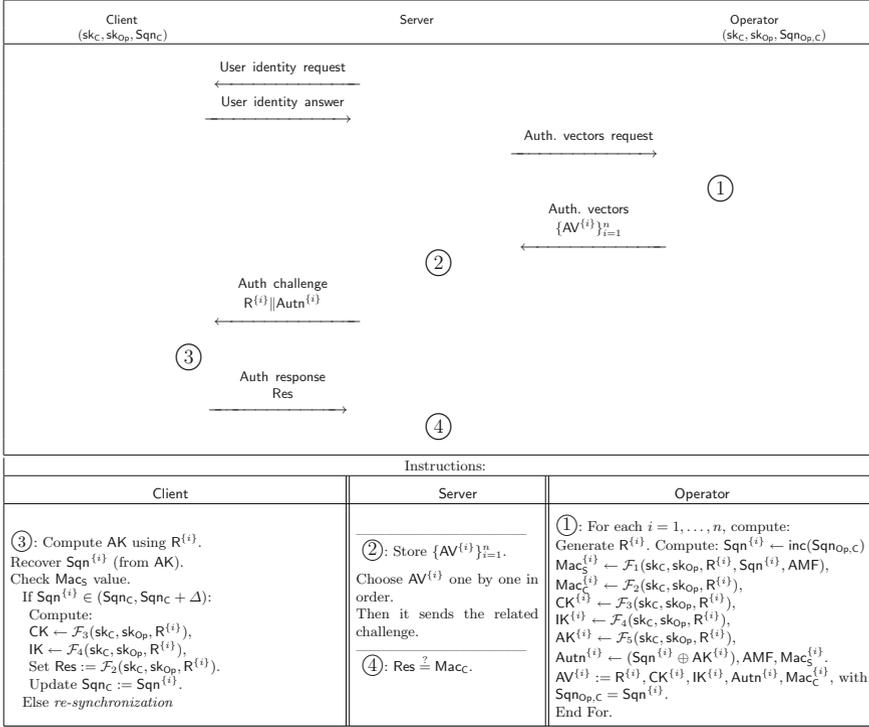


Fig. 1. The AKA procedure.

and  $IK$ ; and a one-time-pad encryption of  $Sqn_{Op,C}$  with a pseudorandom string  $AK$ . The values  $Mac_S, Mac_C, CK, IK, AK$  are output by cryptographic algorithms denoted  $\mathcal{F}_1, \dots, \mathcal{F}_5$  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  $sk_C$ , the operator key  $sk_{Op}$ , 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:  $AV = (R, CK, IK, Mac_S, Mac_C, AMF, AK \oplus Sqn_{Op,C})$ , 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  $Autn = (Sqn_{Op,C} \oplus AK) \parallel AMF \parallel Mac_S$  as a challenge. The client uses  $R$  to compute  $AK$  and recover  $Sqn_{Op,C}$ . If the received  $Mac_S$  verifies and  $Sqn_{Op,C}$  is within a predefined distance  $\Delta$  of  $Sqn_C$ , then  $C$  computes  $(CK, IK)$  and the value  $Mac_C$ , 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  $Sqn_C$  to  $Sqn_{Op,C}$ , and  $S$  verifies the received authentication value with respect to the  $Mac_C$  sent by  $Op$ . If  $Mac_C$  verifies, then  $S$  sends an acknowledgement to  $Op$  and runs a *TMSI re-allocation*. During the optional *re-synchronization*, the client uses  $Sqn_C$

to compute values  $\text{Mac}_S^*$  and  $\text{AK}^* \oplus \text{Sqn}_C$  as  $\text{Op}$  did, using the session  $R$ , but algorithms  $\mathcal{F}_1^*$  and  $\mathcal{F}_5^*$  (not  $\mathcal{F}_1$  and  $\mathcal{F}_5$ ). If  $\text{Mac}_S^*$  verifies,  $\text{Op}$  resets  $\text{Sqn}_{\text{Op},C}$  to  $\text{Sqn}_C$  and sends to  $S$  another batch of  $\text{AV}$  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  $\text{CK}$ . 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  $\text{TMSI}_{\text{new}}$ , is only permanently saved by  $S$  if acknowledged by the client; else, both  $\text{TMSI}_{\text{new}}$  and  $\text{TMSI}_{\text{old}}$  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  $\text{UID}$ , which we consider public. In practice,  $\text{UID}$  is the user’s  $\text{IMSI}$ , which is used in case a TMSI value is not traceable to an  $\text{IMSI}$ . 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  $\text{IMSI}$  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  $\text{MiM}$ ), then we could prove a stronger degree of server impersonation than we currently do. As discussed in Sect. 3, an active  $\text{MiM}$  can inject false TMSI values, which make servers request an  $\text{IMSI}$  value; if the  $\text{MiM}$  reuses this value, it can impersonate servers by offline relays. The use of the TMSI in AKA is undone by using  $\text{IMSI}$ s as a backup for TMSI’s; also, insecurities in using TMSI’s translate to the identification by  $\text{IMSI}$ .

### 3 Security Model

In this section, we propose a security model with respect to two types of adversaries: an active  $\text{MiM}$  with access to the insecure channel between the client and the server; and a malicious server, which also has access to operators. Our security notions are: key-indistinguishability for the session keys  $\text{CK}$ ,  $\text{IK}$ , and client- and server-impersonation resistance. With respect to servers, we also require the key-confidentiality of the client’s long term data  $\text{sk}_C, \text{sk}_{\text{Op}}, \text{Sqn}_C$ , and soundness. We use similar oracles for all the definitions. While we cannot use a basic BPR syntax [7] in this three-party setting, we guarantee a same kind of security with respect to  $\text{MiM}$  attackers. While our server-impersonation model is slightly weaker than that for client-impersonation, this has no impact on the key-indistinguishability for the session keys. For clarity, we include here only intuitive description of the oracles, and leave the formalization for the full version.

**Set Up and Participants.** We consider a set  $\mathcal{P}$  of participants, which are either a server  $S_i$  or a mobile client  $C_i$ . Operators  $\text{Op}$  are not modelled as active parties; in all security games with respect to  $\text{MiM}$  adversaries, operators are black-box algorithms within each server  $S_i$ . For security with respect servers, the operators

are oracles which malicious servers may query. We assume there are  $n_C$  clients,  $n_S$  servers, and  $n_{Op}$  operators. For MiM models, servers contain “copies” of all operators; the copies are assumed to be synchronized with respect to client state, though their output might depend on which server queries them. We associate each client with: a unique identifier  $UID$ , long-term static keys  $(sk_{UID}, sk_{Op})$ , and an ephemeral state  $st_{UID}$  which is a sequence number  $Sqn_{UID}$ . Each of the  $n_S$  servers has black-box access to operator algorithms (or oracles for state-confidentiality and soundness)  $Op_1, \dots, Op_{n_{Op}}$ , initialised with long-term keys  $(sk_{Op_i})$  and tuples  $(UID, sk_C, Sqn_{Op,C})$ , the last value dynamically updated. For simplicity, we assume that the key space of all operators is identical.

**Security Against MiM Adversaries.** In our model, the clients and servers may run concurrent executions of the protocol  $\Pi$ . We denote the  $j$ -th execution of the protocol by party  $P$  as  $P_j$ , associated with a session ID  $sid$ , a partner ID  $pid$  (consisting either of one or of multiple elements), and an accept/reject bit  $accept$  (explained in detail in Sect. 4). In this case  $P_j$  is a handle, used by a MiM adversary  $\mathcal{A}$  to access the oracles below so as to schedule message deliveries, send tampered messages, or interact arbitrarily with any party. We also use a function  $G$ , which we model as a PRF. For a more detailed description, see our full version.

- $CreateCl(Op)$ : creates a new user  $C$  associated with a unique identifier  $UID$ , a key  $sk_{UID}$  drawn independently and uniformly at random from a key space  $\mathcal{S}$ , the key  $sk_{Op}$  of operator  $Op$ , and a sequence number  $Sqn$  stored in  $st_{UID}$ . The adversary is given  $UID$  and  $Sqn_{UID}$ . The operator  $Op$  is given  $(UID, sk_{UID}, Op, Sqn_{UID})$ , and initializes  $st_{Op,UID} := Sqn_{UID}$ , saving the entry  $(UID, sk_{UID}, sk_{Op}, st_{Op,UID})$  in its database.
- $NewInstance(P)$ : instantiates a new instance of  $\Pi$  for party  $P$ , thus creating the handle  $P_j$ , which is made available to the adversary.
- $Execute(P, i, P', j)$ : simulates an execution of  $\Pi$  between the initiating instance  $P_i$  and the instance  $P'_j$  outputting the transcript  $\tau$ .
- $Send(P, i, m)$ : sends message  $m$  to instance  $P_i$ , which outputs a response  $m'$ .
- $Reveal(P, i)$ : outputs the session key(s)  $K$  of instance  $P_i$ .
- $Corrupt(P)$ : If  $P$  is a client, return the key  $sk_C$ , but not  $sk_{Op}$ <sup>6</sup>. If  $P$  is a server, return  $sk_{Op}$ , giving the adversary access to oracle  $OpAccess$ . Corrupted parties become *adversarially controlled*.
- $OpAccess(S, C)$ : gives the adversary access to the local copy of all the operators stored “inside” a corrupted server  $S$ ; the adversary receives as output the message  $Op$  returns if  $S$  queries  $Op$  concerning a client  $C$ .

<sup>6</sup> In this we keep faithful to the implementation of AKA, which protects  $sk_{Op}$  from the user by storing a 1-way function of  $sk_{Op}$  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  $sk_{Op}$ , with no security loss.

- $\text{StReveal}(C, i, \text{bit}_5)$ : returns the state of a client  $C_i$  if  $\text{bit}_5 = 0$  or the state of an operator with respect to a client if  $\text{bit}_5 = 1$ .

We consider two classes of adversaries  $\mathcal{A}$ , *weak* and *strong*, depending on whether  $\mathcal{A}$  may corrupt servers or not. We model three requirements with respect to MiM adversaries.

The notion of *key indistinguishability* demands that the session keys for each execution be indistinguishable from random bitstrings of equal length. The corresponding game is played as follows. The challenger generates the keys of all the  $n_{\text{Op}}$  operators and the  $n_C$  clients; then it gives the  $n_S$  servers  $S_i$  black-box access to the operators. The adversary may query any of the oracles above, and finally issue a single *Test* query on a fresh instance  $P_i$ , which returns either the real keys this instance computed, or random ones of the same length. Strong adversaries can gain oracle access to the copies of the operators in that server. We say that an instance is *fresh* if, and only if: neither the party, nor the partner is corrupted, and no key-reveal was done either on this party, nor on the partner. We define *partner* instances as having the same *session ID*  $\text{sid}$ , which will consist of a random number  $R$ , the client key  $\text{sk}_C$ , the operator key  $\text{sk}_{\text{Op}}$ , and the sequence number used in the successful server authentication  $\text{Sq}_{\text{Op}, C}$ <sup>7</sup>.

Finally,  $\mathcal{A}$  determines whether the returned keys were real or random, and wins if, and only if its response is correct. The adversary’s advantage is defined as:

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

**Definition 1 [Weak/Strong Key-Indistinguishability].** *A key-agreement protocol  $\Pi$  is  $(t, q_{\text{exec}}, q_{\text{res}}, q_G, \epsilon)$ -weakly key-indistinguishable (resp.  $(t, q_{\text{exec}}, q_{\text{res}}, q_s, q_{\text{Op}}, q_G, \epsilon)$ -strongly-key-indistinguishable) if no adversary running in time  $t$ , creating at most  $q_{\text{exec}}$  party instances with at most  $q_{\text{res}}$  resynchronizations per instance, (corrupting at most  $q_s$  servers and making at most  $q_{\text{Op}}$   $\text{OpAccess}$  queries per operator per corrupted server for strong security), and making at most  $q_G$  queries to function  $G$ , has an advantage  $\text{Adv}_{\Pi}^{\text{K.Ind}}(\mathcal{A}) > \epsilon$ .*

We also consider *impersonation* attacks, in which  $\mathcal{A}$  aims to impersonate a partner of a fresh instance. Again, the game begins by generating keys; then  $\mathcal{A}$  gains access to all the oracles (except server corruption/operator access for weak adversaries). When  $\mathcal{A}$  stops, she *wins* if, and only if, there exists an instance (server-instance  $S_i$  for client-impersonation, client-instance  $C_i$  for the server-impersonation) that ends in an accepting state and is *fresh*, subject to an offline/online relay attack described below. The adversary’s advantage is:

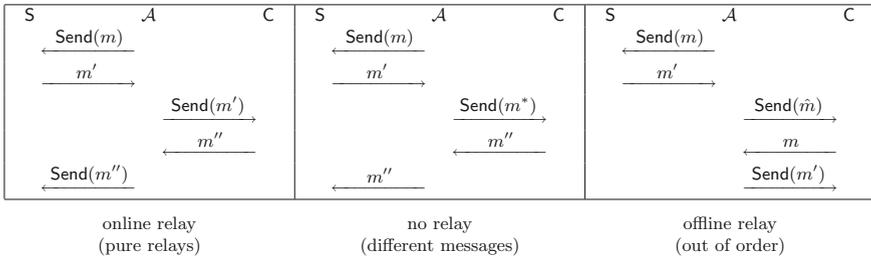
$$\text{Adv}_{\Pi}^{\text{C.Imp}}(\mathcal{A}) := \Pr[\mathcal{A} \text{ wins}], \text{ and respectively } \text{Adv}_{\Pi}^{\text{S.Imp}}(\mathcal{A}) := \Pr[\mathcal{A} \text{ wins}].$$

**Definition 2 [Weak/Strong Impersonation security].** *A key-agreement protocol  $\Pi$  is  $(t, q_{\text{exec}}, q_{\text{res}}, q_G, \epsilon)$ -weak-impersonation-secure (resp.  $(t, q_{\text{exec}}, q_{\text{res}}, q_s,$*

<sup>7</sup> This choice of  $\text{pid}$  and  $\text{sid}$  makes our security guarantee non-composable; however, the design of AKA makes it hard to define  $\text{pids}$  based only on publicly-known values.

$(q_{Op}, q_G, \epsilon)$ -strong-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  $\text{Adv}_{\Pi}^{C.Imp}(\mathcal{A}) \geq \epsilon$  or  $\text{Adv}_{\Pi}^{S.Imp}(\mathcal{A}) \geq \epsilon$ .

Though AKA is claimed to provide mutual authentication, its design introduces a vulnerability, leading to a subtle difference between the *client*-impersonation and *server*-impersonation guarantees. In fact, the protocol allows  $\mathcal{A}$  to run a MiM attack resembling a relay attack. Servers can be impersonated even if we rule out online relays (an adversary just forwards messages from a server to a client instance, and vice versa):  $\mathcal{A}$  merely performs an out-of-order (offline) relay as described in the third scenario of Fig. 2, as explained below. This is the gap between the client- and the server-impersonation guarantees for the AKA protocol. Our server-impersonation model rules out both offline and online relays, whereas client-impersonation only rules out online relays.



**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.

**Security Against Servers.** We also formalize the notions of *state-confidentiality* and *soundness* with respect to a malicious server  $S$ . The former requirement demands that (malicious) servers cannot learn the values:  $sk_C$ ,  $sk_{Op}$ , and the tuple  $(Sqn_C, Sqn_{Op,C})$ . We use a similar model as for the MiM-adversary model, except that now the adversary has oracle access to the operators. We preserve the oracles UReg, NewInstance, Execute, Send, Reveal, StReveal described above, and add the following two oracles (with a modification of Corrupt):

- **Corrupt(P)**  $\rightarrow S$ : if  $P$  is a client, behave as in the MiM model. If  $P$  is an operator, return  $sk_{Op}$  and the tuples  $S = (UID, sk_{UID}, st_C, st_{Op,C})$  of all clients  $C$  subscribing to  $Op$ .
- **OpAccess(S, C)**  $\rightarrow m$ : simulates querying  $C$ ’s operator on behalf of  $C$  for a single session, returning the message  $m$  that  $Op$  outputs to a corrupted  $S$ .

As opposed to key-indistinguishability, which guarantees the security of the session keys, state confidentiality protects the client- and operator long-term states against malicious servers. The state-confidentiality game begins by generating client- and operator material. The adversary can use her oracles arbitrarily, finally outputting a tuple:  $(P_i, sk_{UID}^*, sk_{Op}^*, st_{UID}^*, st_{Op,UID}^*)$  for an uncorrupted client with identifier  $UID$  such that no partner of any instance of  $P$  has ever been corrupted. We say  $\mathcal{A}$  and wins if at least one of the values:  $sk_{UID}^*, sk_{Op}^*, st_{UID}^*, st_{Op,UID}^*$  is equal to the client's real  $sk_{UID}, sk_{Op}, st_{UID}, st_{Op,UID}$ . The advantage of  $\mathcal{A}$  is:  $Adv_{\Pi}^{St.Conf}(\mathcal{A}) := \Pr[\mathcal{A} \text{ wins}]$ .

**Definition 3 [State-confidentiality].** *A key-agreement protocol  $\Pi$  is  $(t, q_{exec}, q_{res}, q_{Op}, q_G, \epsilon)$ -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  $q_{Op}$  `OpAccess` queries and  $q_G$  queries to  $G$ , has an advantage  $Adv_{\Pi}^{St.Conf}(\mathcal{A}) \geq \epsilon$ .*

We also require the property of *soundness*, which demands that malicious servers cannot make an uncorrupted client instance terminate in an accepting state without help from the operator. This game resembles impersonation-security, but the adversary is now a legitimate server with operator access, interacting with the state-confidentiality oracles arbitrarily, making  $q_{Op}$  `OpAccess` queries per client. The adversary wins if, and only if, there exist  $(q_{Op} + 1)$  uncorrupted client instances that terminate in an accepted state. We also restrict this notion with respect to offline replay attacks, as for server-impersonation. The advantage of  $\mathcal{A}$  is defined as:  $Adv_{\Pi}^{Sound}(\mathcal{A}) := \Pr[\mathcal{A} \text{ wins}]$ .

**Definition 4 [Soundness].** *A key-agreement protocol  $\Pi$  is  $(t, q_{exec}, q_{res}, q_{Op}, q_G, \epsilon)$ -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  $Adv_{\Pi}^{Sound}(\mathcal{A}) \geq \epsilon$ .*

## 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 Sect. 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 authenticates 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  $(UID, R, Sqn_{Op,C})$ . During re-synchronization, the operator updates  $Sqn_{Op,C}$  to the client's  $Sqn_C$ ; this update is taken into account in the  $sid$ . Any two partners (same  $sid$ ) with accepting states compute session keys  $(CK||IK)$ .

**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  $\text{sk}_{\text{Op}}$  rather than the client key  $\text{sk}$ . 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  $\Pi$  the AKA protocol described in Sect. 2, but in which the calls to the internal cryptographic functions  $\mathcal{F}_1, \dots, \mathcal{F}_5, \mathcal{F}_1^*, \mathcal{F}_5^*$  are replaced by calls to the function  $G : \{0, 1\}^\kappa \times \{0, 1\}^d \times \{0, 1\}^t \times \{0, 1\}^t \rightarrow \{0, 1\}^n$ , in which  $\kappa$  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  $\mathcal{S}_{\text{C}} := \{0, 1\}^\kappa$  the key-space for the client keys and by  $\mathcal{S}_{\text{Op}} := \{0, 1\}^e$ , the key space for operator keys, for some specified  $e < d$  (in practice  $e = 256$ ). Our system features  $n_{\text{C}}$  clients,  $n_{\text{S}}$  servers and  $n_{\text{Op}}$  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 defer the proofs for the full version.

Our security statements are phrased with respect to an adversary  $\mathcal{A}$  trying to break (in some way) the security of  $\Pi$ , which runs in time  $t$ , creates at most  $q_{\text{exec}}$  party instances with at most  $q_{\text{res}}$  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  $q_{\text{s}}$  servers and make at most  $q_{\text{Op}}$  OpAccess queries per operator per corrupted server. For the *legitimate-and-malicious* adversary, we quantify  $\mathcal{A}$  in terms of the maximal number  $q_{\text{Op}}$  of queries to the oracle OpAccess, and the similar  $q_{\text{exec}}$ ,  $q_{\text{res}}$  and  $q_G$  queries.

The function  $G$  is defined as above.

**Theorem 1** [W.K.Ind, S.C.Imp, W.S.Imp, Sound]. *For the protocol  $\Pi$  using the unitary function  $G$  described above, the following properties hold:*

**W.K.Ind.** *For any  $(t, q_{\text{exec}}, q_{\text{res}}, q_G)$ -adversary  $\mathcal{A}$  against the W.K.Ind-security of  $\Pi$  winning with advantage  $\text{Adv}_{\Pi}^{\text{W.K.Ind}}(\mathcal{A})$  there exists a  $(t' \approx O(t), q' = q_G + q_{\text{exec}}(2 + q_{\text{res}}))$ -adversary  $\mathcal{A}'$  against the pseudorandomness of  $G$  with:*

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

**S.C.Imp.** *For any  $(t, q_{\text{exec}}, q_{\text{res}}, q_s, q_{\text{Op}}, q_G)$ -adversary  $\mathcal{A}$  against the S.C.Imp-security of  $\Pi$ , winning with advantage  $\text{Adv}_{\Pi}^{\text{S.C.Imp}}(\mathcal{A})$ , there exists a  $(t' \approx O(t), q' = 5 \cdot q_s \cdot q_{\text{Op}} + q_G + q_{\text{exec}}(q_{\text{res}} + 2))$ -adversary  $\mathcal{A}'$  against the pseudorandomness of  $G$  such that:*

$$\text{Adv}_{\Pi}^{\text{S.C.Imp}}(\mathcal{A}) \leq n_C \cdot \left( 2 \cdot \text{Adv}_G^{\text{prf}}(\mathcal{A}') + \frac{(q_{\text{exec}} + q_s \cdot q_{\text{Op}})^2}{2^{|\mathbf{R}|}} + \frac{q_{\text{exec}} \cdot q_{\text{res}}}{2^{|\text{Res}|}} + \frac{1}{2^{\kappa}} \right).$$

**W.S.Imp.** *For any  $(t, q_{\text{exec}}, q_{\text{res}}, q_G)$ -adversary  $\mathcal{A}$  against the W.S.Imp-security of  $\Pi$ , winning with advantage  $\text{Adv}_{\Pi}^{\text{W.S.Imp}}(\mathcal{A})$ , there exists a  $(t' \approx t, q = q_{\text{exec}} \cdot (q_{\text{res}} + 2) + q_G)$ -adversary  $\mathcal{A}'$  against the pseudorandomness of  $G$  such that:*

$$\text{Adv}_{\Pi}^{\text{W.S.Imp}}(\mathcal{A}) \leq n_C \cdot \left( \text{Adv}_G^{\text{prf}}(\mathcal{A}') + \frac{q_{\text{exec}} \cdot q_{\text{res}}}{2^{|\text{Mac}_s|}} + \frac{1}{2^{\kappa}} \right).$$

**Sound.** *For any  $(t, q_{\text{exec}}, q_{\text{res}}, q_{\text{Op}}, q_G, \epsilon)$ -adversary  $\mathcal{A}$  against the soundness of  $\Pi$ , winning with advantage  $\text{Adv}_{\Pi}^{\text{Sound}}(\mathcal{A})$ , there exists a  $(t' \approx t, q' = 5 \cdot q_{\text{Op}} + q_G + n_C \cdot q_{\text{exec}}(2 + q_{\text{res}}))$ -adversary  $\mathcal{A}'$  against the pseudorandomness of  $G$  such that:*

$$\text{Adv}_{\Pi}^{\text{Sound}}(\mathcal{A}) \leq n_C \cdot \left( 2 \cdot \text{Adv}_G^{\text{prf}}(\mathcal{A}') + \frac{q_{\text{exec}} \cdot q_{\text{res}}}{2^{|\text{Mac}_s|}} + \frac{1}{2^{\kappa}} \right).$$

**Theorem 2** [St.Conf-resistance]. *For the protocol  $\Pi$  using the unitary functions  $G, G^*$ , for any  $(t, q_{\text{exec}}, q_{\text{res}}, q_{\text{Op}}, q_G, q_{G^*})$ -adversary  $\mathcal{A}$  against the St.Conf-security of  $\Pi$ , winning with advantage  $\text{Adv}_{\Pi}^{\text{St.Conf}}(\mathcal{A})$ , there exist: a  $(t' \approx O(t), q' = q_G + q_{\text{exec}}(5 + q_{\text{res}}))$ -prf-adversary  $\mathcal{A}_1$  on  $G$  and  $(t' \approx O(t), q' = q_{G^*})$ -prf-adversary  $\mathcal{A}_2$  on  $G^*$  such that:*

$$\text{Adv}_{\Pi}^{\text{St.Conf}}(\mathcal{A}) \leq n_C \cdot \left( \frac{1}{2^{|\text{sk}_c|}} + \frac{1}{2^{|\text{sk}_{\text{Op}}|}} + \frac{2}{2^{|\text{sqn}|}} + \text{Adv}_G^{\text{prf}}(\mathcal{A}_1) + \text{Adv}_{G^*}^{\text{prf}}(\mathcal{A}_2) \right).$$

**MILENAGE and TUAK.** Our second step is to prove that TUAK and MILENAGE both behave 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 full paper. 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  $\text{sk}_{\text{Op}}$ ).

**Theorem 3 [prf-security for TUAK algorithms].** *For the generalization of the TUAK algorithms  $G_{\text{tuak}}$  (resp.  $G_{\text{tuak}}^*$ ) 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:*

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

**Theorem 4 [prf-security for MILENAGE algorithms].** *For the generalization of the MILENAGE algorithms  $G_{\text{mil1}}$  and  $G_{\text{mil2}}$  (resp.  $G_{\text{mil1}}^*$  and  $G_{\text{mil2}}^*$ ) 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:*

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

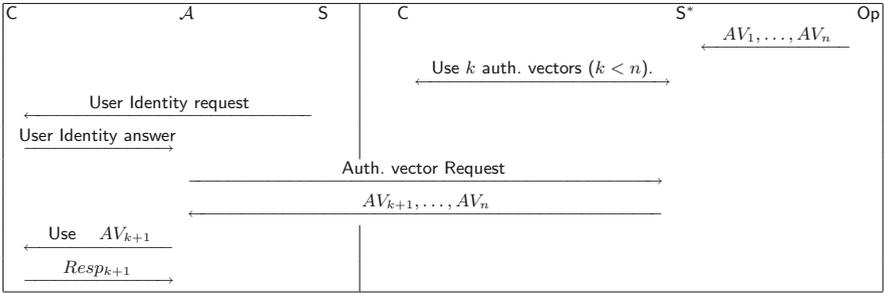
## 4.2 Vulnerabilities of the AKA Protocol

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 [15] and Zhang and Fang [16] 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.

The main attack strategy is also depicted in Fig. 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  $(AV_1, \dots, AV_n)$ , using some of them (vectors  $AV_1, \dots, AV_k$ ) 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  $AV_{k+1}, \dots, AV_n$ ; 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



**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  $AV_1, \dots, AV_k$ . Then, the server  $S^*$  is corrupted, and the adversary  $\mathcal{A}$  learns  $AV_{k+1}, \dots, AV_n$ , which it uses in a second attack phase (on the left).

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  $Id_S$ , to the input of each of the cryptographic functions, thus making any leftover authentication tokens un-replayable in the wrong area. We stress that this is a minor modification to the protocol, as servers are already associated with a unique LAI identifier.

We also show in the full version how to include  $Id_S$  in the computation of each of the cryptographic algorithms. We present our modified protocol in Fig. 4.

**Security of the Modified AKA Protocol.** This modification still (trivially) preserves the properties of strong client-impersonation resistance, soundness, and state confidentiality. However, the modification yields in addition strong key-indistinguishability and server-impersonation resistance, as we detail below. The proofs are given in the full version.

**Theorem 5** [S.K.Ind, S.S.Imp]. *For the modified protocol  $\Pi$  using the unitary function  $G$  described in Sect. 4, the following properties also hold:*

S.K.Ind. *For any  $(t, q_{exec}, q_{res}, q_s, q_{Op}, q_G)$ -adversary  $\mathcal{A}$  against the S.K.Ind-security of  $\Pi$  winning with advantage  $Adv_{\Pi}^{S.K.Ind}(\mathcal{A})$  there exists a  $(t' \approx$*

Instructions:		
Client	Server	Operator
<p>③: Compute AK using <math>R^{(i)}</math>. Recover <math>Sq_n^{(i)}</math> (from AK). Check Macs value. If <math>Sq_n^{(i)} \in \{Sq_{nC}, Sq_{nC} + \Delta\}</math>: Compute: CK <math>\leftarrow \text{Upd.F}_3(sk_C, sk_{Op}, R^{(i)}, Id_S)</math>, IK <math>\leftarrow \text{Upd.F}_4(sk_C, sk_{Op}, R^{(i)}, Id_S)</math>, Set <math>Res := \text{Upd.F}_2(sk_C, sk_{Op}, R^{(i)}, Id_S)</math>. Update <math>Sq_{nC} := Sq_n^{(i)}</math>. Else re-synchronization</p>	<p>②: Store <math>\{AV^{(i)}\}_{i=1}^n</math>. Choose <math>AV^{(i)}</math> one by one in order. Then, it forges and sends the related challenge. ④: <math>Res \stackrel{?}{=} Mac_{cc}</math>.</p>	<p>①: For each <math>i = 1, \dots, n</math>, compute: Generate <math>R^{(i)}</math>. Compute: <math>Sq_n^{(i)} \leftarrow \text{inc}(Sq_{nOp,C})</math> <math>Mac_c^{(i)} \leftarrow \text{Upd.F}_1(sk_C, sk_{Op}, R^{(i)}, Sq_n^{(i)}, AMF, Id_S)</math>, <math>Mac_c^{(i)} \leftarrow \text{Upd.F}_2(sk_C, sk_{Op}, R^{(i)}, Id_S)</math>, <math>CK^{(i)} \leftarrow \text{Upd.F}_3(sk_C, sk_{Op}, R^{(i)}, Id_S)</math>, <math>IK^{(i)} \leftarrow \text{Upd.F}_4(sk_C, sk_{Op}, R^{(i)}, Id_S)</math>, <math>AK^{(i)} \leftarrow \text{Upd.F}_5(sk_C, sk_{Op}, R^{(i)}, Id_S)</math>, <math>Autn^{(i)} \leftarrow (Sq_n^{(i)} \oplus AK)</math>, AMF, Macs. <math>AV^{(i)} := (R^{(i)}, CK^{(i)}, IK^{(i)}, Autn^{(i)}, Mac_c^{(i)})</math>, with <math>Sq_{nOp,C} = Sq_n^{(i)}</math>. End For.</p>

Fig. 4. The modified instructions of our variant.

$O(t), q' = 5 \cdot q_s \cdot q_{Op} + q_G + q_{exec}(q_{res} + 2)$ -adversary  $\mathcal{A}'$  against the pseudo-randomness of  $G$  with:

$$\text{Adv}_{\Pi}^{\text{S.K.Ind}}(\mathcal{A}) \leq n_C \cdot \left( \frac{(q_{exec} + q_s \cdot q_{Op})^2}{2^{|R|}} + 2 \cdot \text{Adv}_G^{\text{prf}}(\mathcal{A}') \right).$$

S.S.Imp. For any  $(t, q_{exec}, q_{res}, q_s, q_{Op}, q_G)$ -adversary  $\mathcal{A}$  against the S.S.Imp-security of  $\Pi$ , winning with advantage  $\text{Adv}_{\Pi}^{\text{S.S.Imp}}(\mathcal{A})$ , there exists a  $(t' \approx O(t), q' = 5 \cdot q_s \cdot q_{Op} + q_G + q_{exec}(2 + q_{res}))$ -adversary  $\mathcal{A}'$  against the pseudo-randomness of  $G$  such that:

$$\text{Adv}_{\Pi}^{\text{S.S.Imp}}(\mathcal{A}_{G_0}) \leq n_C \cdot \left( \frac{q_{exec} \cdot q_{res}}{2^{|Macs|}} + \frac{1}{2^{\kappa}} + 2 \cdot \text{Adv}_G^{\text{prf}}(\mathcal{A}') \right).$$

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

## References

- 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
- 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
- Shaik, A., Borgaonkar, R., Asokan, N., Niemi, V., Seifert, J.-P.: Practical attacks against privacy and availability in 4G/LTE mobile communication systems. In: Accepted to NDSS 2016 (2016)

4. Arapinis, M., Chothia, T., Ritter, E., Ryan, M.: Analysing unlinkability and anonymity using the applied Pi calculus. In: Proceedings of the CSF 2010, pp. 107–121 (2010)
5. Arapinis, M., Mancini, L.I., Ritter, E., Ryan, M., Golde, N., Redon, K., Borgaonkar, R.: New privacy issues in mobile telephony: fix and verification. In: Proceedings of ACM CCS
6. Arapinis, M., Ritter, E., Ryan, M.D.: StatVerif: verification of stateful processes. In: Proceedings of CSF 2011, pp. 33–47 (2011)
7. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
8. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994)
9. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the indistinguishability of the sponge construction. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 181–197. Springer, Heidelberg (2008)
10. Blanchet, B.: Automatic verification of security protocols in the symbolic model: the verifier ProVerif. In: Aldini, A., Lopez, J., Martinelli, F. (eds.) FOSAD VII. LNCS, vol. 8604, pp. 54–87. Springer, Heidelberg (2014)
11. Gilbert, H.: The security of “One-Block-to-Many” modes of operation. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 376–395. Springer, Heidelberg (2003)
12. Hall, C., Wagner, D., Kelsey, J., Schneier, B.: Building PRFs from PRPs
13. Lee, M., Smart, N., Warinschi, B., Watson, G.: Anonymity guarantees of the UMTS/LTE authentication and connection protocol. *Int. J. Inf. Sec.* **13**(6), 513–527 (2014)
14. Strobel, D.: IMSI catcher. In: Seminar Work, Ruhr-Universität Bochum (2007)
15. Zhang, M.: Provably-Secure Enhancement on 3Gpp. Authentication and Key Agreement Protocol. In: IACR Cryptology ePrint Archive 2003, p. 92 (2003). <http://eprint.iacr.org/2003/092>
16. Zhang, M., Fang, Y.: Security analysis and enhancements of 3GPP authentication and key agreement protocol. *IEEE Trans. Wirel. Commun.* **4**(2), 734–742 (2005)



<http://www.springer.com/978-3-319-39554-8>

Applied Cryptography and Network Security  
14th International Conference, ACNS 2016, Guildford,  
UK, June 19-22, 2016. Proceedings  
Manulis, M.; Sadeghi, A.-R.; Schneider, S. (Eds.)  
2016, XIV, 668 p. 110 illus., Softcover  
ISBN: 978-3-319-39554-8