Achieving Better Privacy for the 3GPP AKA Protocol

Pierre-Alain Fouque[1], Cristina Onete[2] and Benjamin Richard[3]

[1] Université de Rennes 1/IRISA France, pa.fouque@gmail.com
[2] INSA Rennes / IRISA, France, cristina.onete@gmail.com
[3] Orange Labs, Chatillon, France , benjamin.richard@orange.com

Abstract. Proposed by the 3rd Generation Partnership Project (3GPP) as a standard for 3G and 4G mobile-network communications, the AKA protocol is meant to provide a mutually-authenticated key-exchange between *clients* and associated network *servers*. As a result AKA must guarantee the indistinguishability from random of the session keys (key-indistinguishability), as well as client- and server-impersonation resistance. A paramount requirement is also that of client privacy, which 3GPP defines in terms of: *user identity confidentiality*, *service untraceability*, and *location untraceability*. Moreover, since servers are sometimes untrusted (in the case of roaming), the AKA protocol must also protect clients with respect to these third parties. Following the description of client-tracking attacks e.g. by using error messages or IMSI catchers, van den Broek *et al.* and respectively Arapinis *et al.* each proposed a new variant of AKA, addressing such problems. In this paper we use the approach of provable security to show that these variants still fail to guarantee the privacy of mobile clients. We propose an improvement of AKA, which retains most of its structure and respects practical necessities such as key-management, but which provably attains security with respect to servers and Man-in-the-Middle (MiM) adversaries. Moreover, it is impossible to link client sessions in the absence of client-corruptions. Finally, we prove that any variant of AKA retaining its mutual authentication specificities *cannot* achieve client-unlinkability in the presence of corruptions. In this sense, our proposed variant is optimal.

Keywords: privacy, security proof, AKA protocol

1 Introduction

Authenticated key-exchange (AKE) protocols allow two parties to communicate securely over an insecure channel, such as the Internet, a radio-frequency (RF) channel, or a mobile telecommunications network. In general, AKE protocols consist of two parts: (1) an authenticated *key-exchange procedure*, which allows two parties to derive a set of session keys; (2) a *record layer protocol*, allowing the two parties to exchange confidential, authentic data by using the derived keys. Such protocols are essential to ensure that sensitive data or services are securely provided from to a legitimate client.

In the context of mobile networks, mobile clients subscribe to an operator, and consequently become entitled to specific services. Whenever the clients then require a specific service, they run an authenticated key-exchange protocol called AKA with a location-specific server, which then provides the service.

The AKA protocol is executed between three entities: the mobile clients, the server and the operator. The second entity is trusted to run the AKA protocol with the clients, bit are not entitled to know the client's, nor the operator's secret values. In 3GPP syntax, the server is the Visted Location Register (VLR) and the operator is the related Home Location Register (HLR). The AKA protocol was introduced by the Third Generation Partnership Project (3GPP), which wrote and has been maintaining the specifications of 3G telecommunication systems. The protocol relies on a suite of seven cryptographic algorithms. Two proposals for the latter exist: one relies on AES (the suite is called MILENAGE) and another relies on the Keccak hash function (this suite is called TUAK).

The minimal security requirement for this scenario is ensuring that no mobile client illicitly gains access to mobile services without being entitled to it. Similarly, we require that no attacker can make another client pay for mobile services they are not using.

An important additional concern is that of client privacy with respect to third-party outsiders; recent concerns of illicit eavesdropping and mass-surveillance especially indicate that third parties should not be able to identify, nor trace, mobile clients. The AKA protocol explicitly requires three flavors of user privacy [3]: user identity confidentiality, user untraceability, and user location confidentiality. The first of these notions demands that an adversary that is eavesdropping on radio access links cannot recover the permanent identifier associated to each client (called an IMSI). This notion is specific to the AKA protocol, and it is somewhat limited, since users may also be tracked by learning another user-specific value, namely the sequence number. In particular, protecting only the permanent identifier does not enforce a very strong notion of privacy. The second notion of privacy required by the 3GPP specifications is that of user untraceability, which refers to an attacker's ability of learning whether multiple services were used by the same user. Finally, the third notion is specific to mobile networks, requiring that external attackers cannot learn the area where the victim clients are when they run the AKA protocol (since the servers running AKA are area specific).

In order to deliver services to mobile clients, these users run the AKA protocol with a local server, specific to a given area; if mutual authentication succeeds (i.e. both parties ensure the legitimacy of their conversation partner), then both parties derive a set of keys. Then, the client and server both use the obtained session keys to ensure the confidentiality and authenticity of any further data exchanged between them (in particular, any mobile service required by the client).

The 3GPP AKA protocol uses a symmetric-key infrastructure and several sensitive parameters to identify legitimate clients. Most secret information is shared between the client and its operator, but not with any of the local-area servers running the AKE protocol with the client. Each user is associated with three static values: a permanent identifier IMSI, a secret client key sk, and an operator key $\mathsf{sk}_{\mathsf{op}}$; and two ephemeral values, updated at every protocol run: a temporary identifier TMSI and a sequence number Sqn. Note that, while all the clients subscribing to the same operator share its operator key $\mathsf{sk}_{\mathsf{op}}$, this key is not stored in clear on the client's smartphone; instead, the users store a value Top_C that is pseudorandom, depending on sk and $\mathsf{sk}_{\mathsf{op}}$: thus, two different clients sharing a same operator key, the related values Top_C they store are different and unlinkable. Moreover, note that the TMSI value is shared exclusively between clients and *servers* (not operators), and they are sent by servers to the clients across a channel that is secured by using the derived session keys. Each TMSI value is unique per server, and since each server is associated with a single area (as denoted by a Local Area Identifier – LAI), the tuple consisting of the TMSI and the relevant LAI is unique per client.

The AKA protocol was designed to guarantee the three notions of privacy outlined before. The permanent identifier IMSI of each client is not meant to be used in regular protocol runs; instead, the unique tuple of a TMSI and the identifier LAI of the area that the protocol was last used from will uniquely identify a client. Services are transmitted across the secure channel; thus, the combined security of the derived session keys and of the record-layer algorithms, and the privacy of the client running the authenticated key-exchange protocol are supposed to guarantee client untraceability. Finally, by hiding the *current* location area identifier, one ensures that only the client's *immediate, past* location is revealed.

There are three distinct areas that our work is related to: previous results concerning the privacy of the AKA protocol; provable security results in (symmetric-key) privacy; related work on key-exchange and secure-channel establishment protocols. We review these and state our contributions in the following.

The Privacy of AKA. Though initially the designers of AKA provided a security proof using BAN logic [11], subsequent vulnerabilities belied these security guarantees. The attacks of [24] and [26] indicate that servers can actually be impersonated within the protocol run; the second paper also indicates that well-known weaknesses of the GSM protocol, such as the use of weak encryption and the lack of mutual authentication, can pose security problems for AKA. Zhang and Fang [18] pointed out that the use of sequence numbers, together with potential corrupted server redirection, allow attackers to illicitly trace and impersonate clients.

A known *privacy* problem of the AKA protocol is the "IMSI catcher" attack [6], which exploits the fact that the permanent identifier IMSI is sent as a back-up for a faulty tuple $(\mathsf{TMSI}, \mathsf{LAI})$. By either observing such faulty behaviour (due to transmission errors or to server database problems), or by causing it, MiM

attackers can easily track clients, in blatant violation of the user identity confidentiality requirement. This is a problem in all generations of mobile communication networks.

Very recently, van den Broek et al. [7] proposed a countermeasure to IMSI catchers, which essentially replaces IMSI values by an unlinkable pseudonym, denoted PMSI. Aside from any practical issues, such as requiring the operator's presence at every protocol run, we show that an active adversary can still easily desynchronize the PMSI value, and then track clients. As a second drawback, [7] neither specifies, nor addresses any other threats to privacy.

Arapinis et al. [20] showed that failure messages (in authentication/resynchronization) can be used to trace users. A subsequent paper by some of the same authors [19] cleverly identifies ways in which a specific implementation deviates from 3GPP recommendations and thus allow linkability of client sessions.

Though several improvements of AKA have been proposed, only two analyse the security of the resulting construction. A first one [17] describes AP-AKA, a stateless protocol which can thwart replay-based impersonation attacks and lowers the impact of server corruptions, but at the cost of no privacy at all. A second modification [20] retains the stateful nature of AKA, but the proof they provide models the parties' state in an idealized way, considering it to be random. We show weaknesses in both proposals in Section 3.3.

Lee et al. [16] propose no improvement to the protocol itself, but do cryptographically analyse the privacy of AKA, focusing only on the identification phase. Their restricted model takes into account known attacks, but they "merge" servers and operators into a single party. This forcibly implies that the server is assumed to be honest; furthermore, server corruptions are not taken into account. Their PRF agility property seems akin to the generalization we make to the function G, and they prove a weak degree of privacy in the absence of corruptions.

Secure AKE models. Bellare and Rogaway first proposed a security model for AKE mechanisms in [14], also in a symmetric setting. Their framework was later extended with the contribution of Pointcheval [12]. A further model for generic session-oriented protocols was proposed in [15] and extended in [22]. Although we use BPR-methodologies in our analysis, we cannot simply "import" their model, and use a slightly modified version thereof.

Privacy models. The privacy model due to Vaudenay [23] was the first to define several adversarial classes for untraceability in the presence of corruptions. His framework was specifically designed for RFID privacy; his approach, however, features adversarial classes and attacks which are universal in authentication scenarios. In particular, Vaudenay captured adaptive corruptions, and classified adversaries in terms of their behaviour upon corruption. The AKA protocol is all the more adapted to his framework since it uses only symmetric keys, as does RFID authentication.

His framework was later refined and extended by Hermans et al. [9], who used Vaudenay's classification of adversaries according to two criteria: (i) whether the adversary is aware of the result of authentication sessions (wide adversaries) or not (narrow adversaries); (ii) any constraints in the corruption behaviour (the adversaries range from weak –no corruptions– to forward –corruptions "end" the game– and strong). In our work, we use the game-based definition of Hermans et al.

Two important impossibility results indicate that strong privacy *requires* key exchange [23] and that in symmetric-key authentication protocols, if the server authenticates to the client during the protocol, there exists an attack that allows an adversary to distinguish between two clients [21]. In particular, simply dropping the last message of the server to one of two clients, then corrupting the clients, and finally simulating them with the and dropped message lets the adversary distinguish between those clients. This attack, however, tacitly assumes that (a) clients do not "reset" if the authentication session is aborted, or alternatively that (b) the adversary can actually corrupt the clients before they time out.

Our contributions. We have two main contributions. The first is to show that both the current version of AKA protocol and two of the more promising improvements proposed in the literature suffer from several flaws, regarding both client-privacy and the security with respect to the server. Our second main contribution is to present a variant of the AKA which provably guarantees the five properties listed below. As an

additional important result, we also prove that achieving a stronger degree of client-unlinkability (see below) is impossible while the AKA protocol retains its current structure.

- **Key-indistinguishability**: a BPR-like [12] guarantee that the derived session keys are indistinguishable from random values of equal length. The adversary here is an active Man-in-the-Middle (MiM).
- **Impersonation security**: the guarantee that a MiM attacker cannot impersonate either the server (to the client), or the client (to the server).
- **Wide-weak client-unlinkability**: a property guaranteeing that two sessions run by the same client are not linkable with respect to a MiM attacker which may learn whether the server accepts the client's authentication or not, but which cannot corrupt clients to learn their keys.
- **State-confidentiality**: the demand that (malicious) servers cannot learn the client's secret key, the corresponding operator's secret key, nor the client's state (in particular, the sequence number). We assume that the server interacts with both operators and with clients, but we only guarantee security for the authenticated key-exchange protocol (we do not address the record-layer primitives).
- **Soundness**: the demand that (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.

An implicit, but also important contribution is formalizing these five security and privacy properties, thus providing a framework for analyzing this protocol. Considering the ease with which certain active attacks (such as active IMSI catcher attacks) can be performed against the AKA protocol, we also assess its resistance to active attacks. We describe a client-unlinkability attack and then an attack against the soundness of AKA. We also present several attacks against two promising improvements on the AKA protocol in the literature. The precise formalization of our notions furthermore allows us to find a gap in an impossibility result regarding client-unlinkability in the presence of corruptions for mutually authenticated protocols [21], and formulate a different impossibility result, which is more precise, and also more generic in the case of the AKA protocol. Another main contribution of this paper is to extend the narrow-forward-privacy impossibility result for a broader class of protocols including AKA and its variants.

Our improved variant of the protocol mostly retains the symmetric character of the current version. However, we bypass IMSI catcher attacks by never revealing IMSI values, instead sending them encrypted with a public-key IND-CCA-secure encryption scheme. We assume each operator has a PKE key-tuple, and each client stores the public key of the operator. This minimizes key-management problems. We note that, just as for the AKA protocol, we only use the (encrypted) IMSI value as an alternative for the randomly-chosen temporary identifier values TMSI. However, we choose to update the current TMSI value by using authenticated encryption (AES-GCM) as part of the server's authentication message. The PKE scheme is also used when moving from area to area: thus, if the client switches from one server (in a given location) to another (in another location), the TMSI is not used. This allows us to reveal only the current area that the server is in, rather than the client's past location (as is the case for the current version of AKA).

We retain the structure of the authenticated key-exchange part of AKA, using the client- and operator-state to authenticate the two parties and the derived session keys. We show that while this feature of the AKA protocol reamins in use, no client-untraceability can be achieved in the presence of corruptions. By removing the need for re-synchronization, we also implicitly prevent attacks which link client sessions depending on whether or not the re-synchronization procedure is used.

Table 1 compares our proposal to the AKA protocol and to two more promising variants.

2 Privacy model

2.1 3GPP Privacy requirements

AKA Infrastructure. The mobile context for which the AKA protocol was designed contains three entities: (1) clients, which register with operators and are allowed to access a subset of services; (2) operators, which

	Defeating:			Security:				
	Attack n°1	Attack n°2	Attack n°3	Prop. n°1	Prop. n°2	Prop. n°3	Prop. n°4	Prop. n°5
3G AKA	x [7]	x [20]	x	x	x	x	?	?
Arapinis	✓ [20]	✓ [20]	x	✓	x	x	?	✓* [20] [4]
Van Den Broek	✓ [7]	x	✓	✓	x	x	?	?
Our variant	✓	✓	✓	✓	✓	x	✓	✓

Attack n°1: IMSI Catcher § 3.3.	Prop. n°1: Confidentiality of the previous location §2.
Attack n°2: Linkability of failure messages § 3.3.	Prop. n°2: ww-unlink §2.2.
Attack n°3 : Our traceability attack § 3.3.	Prop. n°3: nf-unlink §2.2.
Prop. n°4: State-confidentiality & soundness §2.2.	Prop. n°5: Key-indistinguishability & Client- and Server-impersonation §2.2.

Fig. 1. Comparison between several AKA variants.

know the secret parameters of all their registered clients; and (3) local servers, which are tasked with providing services to mobile clients, but are not trusted to know the clients' personal information. In the AKA literature, opeators are usually called home local registers (HLR), while servers are known as VLR.

The security demands of 3G/4G networks are client-centric, revolving around the following parameters related to mobile clients (users) C:

IMSI : a permanent identifier, unique per customer and highly trackable;
TMSI : a temporary identifier, unique per server, and updated after each successful protocol run;
LAI : a unique local-area identifier per server; client store (TMSI, LAI) tuples whenever a server issues a new TMSI;
$\mathsf{sk_C}$: the client's unique client key;
$\mathsf{sk_{op}}$: the key of the operator C subscribes to;
$\mathsf{Sqn_C}, \mathsf{Sqn_{Op,C}}$: the client's state $\mathsf{Sqn_C}$ has an equivalent operator state $\mathsf{Sqn_{Op,C}}$, which should not be "too far" from the client's state. The state is updated by the client upon authenticating the server (correct verification of the authentication challenge); the server updates state upon authenticating the challenge (verification of the authentication response).

We refer the reader to Section 3.2 for more details about the protocol description.

The identifier and key-management schemes are as follows. Clients may know the permanent value IMSI, the temporary identifiers TMSI and LAI, the keys $\mathsf{sk_C}, \mathsf{sk_{op}}$ and a function of $\mathsf{sk_C}$ and $\mathsf{sk_{op}}$ (they do not store $\mathsf{sk_{op}}$ in clear, as discussed in section 3.2). Operators know the tuple: $(\mathsf{IMSI}, \mathsf{sk_C}, \mathsf{sk_{op}})$. Servers keep track of tuples $(\mathsf{TMSI}, \mathsf{LAI}, \mathsf{IMSI})$. Furthermore, both servers and operators know the sequence number that the operator associated to each client. The clients update their own sequence numbers, which are highly related to the operator's sequence number.

Very notably, servers must both authenticate and exchange session keys with mobile clients, despite not knowing their secret material.

Client Privacy. The Third Generation Partnership Project (3GPP), which designed the AKA protocol in the TS.33.102 specification [3], lists the following privacy requirements:

- **user identity confidentiality:** specifically, *"the property that the permanent user identity (*IMSI*) of a user [...] cannot be eavesdropped on the radio access link."*
- **user untraceability:** namely, *"the property that an intruder cannot deduce whether different services are delivered to the same user by eavesdropping on the radio access link."*
- **user location confidentiality:** in particular, *"the property that the presence or the arrival of a user in a certain area cannot be determined by eavesdropping on the radio access link."*

The requirements quoted above are quite informal; moreover, the nomenclature is confusing, since in the *provable-security* literature, *untraceability* refers to adversaries tracing clients in distinct protocol runs

(rather than it being service-related). We discuss the three requirements below, then formalize them into cryptographic requirements.

User identity confidentiality concerns only the client's *permanent* IMSI value (not, e.g. the client's sequence number) with respect to *passive* attackers (rather than active ones). However, mobile networks are notoriously prone to Man-in-the-Middle (MiM) active attacks like the IMSI catcher [6], which allows a third party (the MiM) to recover a client's IMSI. Another highly-trackable client-specific parameter is the sequence number Sqn, whose updating procedure is very simplistic and its output, predictable even without a secrey key. As a consequence we require the stronger property of *provable unlinkability*, which ensures that even an *active* MiM cannot *link* two AKA protocol runs to the same client.

For user untraceability, no attacker must know whether the same service (i.e. *any* message-exchange over the secure channel) is provided to a client multiple times. From the point of view of provable security, this is equivalent to *key-indistinguishability* if the authenticated-encryption algorithms are assumed to be secure.

User location confidentaility demands that eavesdroppers $\mathcal{A}$ cannot detect the presence of a client in a given area; however, the definition does not specify what information $\mathcal{A}$ links to each client (e.g. the IMSI, the sequence number, etc.). Attackers are aware of the current LAI; the difficulty lies in learning which clients *enter* the area. Unfortunately the AKA protocol always reveals the *past* location of any arriving client, making unique (or rare) itineraries stand out. We formalize a strong degree of location privacy as a part of client-unlinkability.

Our formalizations of *client unlinkability* and *key-indis-tinguishability* consequently guarantee 3GPP's three privacy requirements.

Implicit requirements. As discussed in Section 1, the AKA protocol implicitly addresses security with respect to malicious servers, which are restricted as follows: (1) the servers have no access to the tuple $(\mathsf{sk}_\mathsf{C}, \mathsf{sk}_{\mathsf{op}})$; (2) the (hence necessary) operator-server communication must be minimized in order to minimize costs.

We formulate the following two *implicit* requirements:

- **State-Confidentiality:** Servers must not learn any client-related long-term state.
- **Soundness:** Clients must reject authentication-challenges not explicitly provided by the operator to the server.

2.2 Security models

Due to their orthogonality, it is hard to formalize the notions of key-indistinguishability and client-unlinkability in the same generic framework. One difficulty is the fact that the unlinkability notion requires the adversary to have access to clients without knowing their identities. Following established approaches [23,10], in the unlinkability model, we associate clients with identifiers, or handles, denoted VC% (Virtual Client), and this changes the syntax of the oracles we use. Thus, we differentiate between the model for security (including notions of key-indistinguishability, client- and server-impersonation resistance against a MiM adversary, and state-confidentiality and soundness with respect to malicious servers), and that of client-unlinkability. We use similar oracles, with a slightly different syntax, for the two types of definitions, and thus obtain security guarantees based on traditional Bellare, Pointcheval, and Rogaway models [12].

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 of the type respectively VLR or ME/USIM. By contrast operators Op are not modeled as active parties. In all security games apart from state-confidentiality and soundness with respect to the server, the operators Op are black-box algorithms within the server S; in those two games, the operators are oracles, which the adversary (i.e. the server) may query. We assume the existence of n_C clients, n_S servers, and n_{Op} operators. If the operators are contained within the servers, we assume that all copies of the same operator Op are synchronized at all times. We associate each client with: a long-term, static secret state consisting of a tuple $(\mathsf{sk}_\mathsf{C}, \mathsf{sk}_{\mathsf{op}})$, an ephemeral state st_C consisting of a sequence number Sqn_C, a tuple of a static, permanent identifier IMSI and an ephemeral, temporary identifier TMSI, and finally a tuple of a current, and a past local

area identifier, denoted $\mathsf{past.LAI_P}$ and $\mathsf{curr.LAI_P}$ respectively. Servers are associated with a permanent local area identifier LAI and a unique network identifier $\mathsf{ID_{S_i}}$; they also keep track of a list of tuples $(\mathsf{TMSI}, \mathsf{IMSI})$ associated with clients. Each of the at most n_S servers has black-box access to algorithms (or oracles in the case of state-confidentiality and soundness) $\mathsf{Op}_1, \ldots, \mathsf{Op}_{n_{\mathsf{Op}}}$, which are initialized with long-term keys $(\mathsf{sk}_{\mathsf{Op}_i})$ and keep track of a list of tuples $(\mathsf{IMSI}, \mathsf{sk_C}, \mathsf{Sqn}_{\mathsf{Op,C}})$. In our model, we also assume that the key space of all operators is identical (otherwise it becomes easier to distinguish between clients of different operators).

Client Unlinkability Client-Unlinkability.

Informally, we call a protocol Π client-unlinkable if no adversary can know whether two executions of Π were run by the same, or by two different clients. Two sessions associated with the same client are called *linked*. Following previous works of Vaudenay [23] and Hermans et al. [10], we give the adversary access to a basic left-or-right oracle, which associates an anonymized handle virtual client VC to one of two possible clients (input by the adversary). We extend this framework to account for client mobility, giving the adversary access to a relocation oracle. Consequently, if an attacker can distinguish between clients based on their location, they will win the unlinkability game, which we detail below.

At the onset of this game, the set of clients is empty and the challenger instantiates two lists $\mathcal{L}_{\mathsf{drawn}}$ and $\mathcal{L}_{\mathsf{free}}$. We initialize operators by choosing their secret keys. The adversary can then initialise servers by choosing their locations, and it can create clients to populate the system it attacks. For each newly-created client, the past location $\mathsf{past.LAI_C}$ is set to a special symbol $\perp$ and the current location is adversarially-chosen.

The adversary then interacts with clients by means of several oracles. The lists $\mathcal{L}_{\mathsf{drawn}}$ and $\mathcal{L}_{\mathsf{free}}$ correspond to the two possible states of any one client. Clients can be "drawn" or "free"; at creation, all clients are "free", and they may become "drawn" if used as input to a left-or-right Client-Drawing oracle. In particular, we use a left-or-right Client-Drawing oracle (similar to that in [23,10]), which allows the adversary to interact with one of two clients (the interacting client being chosen depending on a secret bit b). Clients input to the Drawing oracle are moved to the $\mathcal{L}_{\mathsf{drawn}}$ list; further Drawing queries can then be made concurrently as long as the input clients are in the $\mathcal{L}_{\mathsf{free}}$ list[5]. Upon drawing one of two possible clients, the adversary is given a handle on the chosen entity; following the notation of [23], we call this a *virtual* client and we associate it with the handle VC. Virtual clients can then be freed by the adversary (this would remove them from the $\mathcal{L}_{\mathsf{drawn}}$ list and re-add them to the $\mathcal{L}_{\mathsf{free}}$ list). Only free clients can be drawn. This oracle associates a handle to either the left or the right input client, depending on a secret bit b.

The client unlinkability property is defined in terms of the following security experiment $\mathsf{Exp}^{\mathsf{c.unlink}}_{\mathcal{A}}(1^\lambda)$, for a security parameter (in unary) 1^λ.

- The challenger randomly chooses a bit $b \in \{0, 1\}$.
- The adversary may use the oracles below (with restrictions depending on its adversarial class), and the challenger answers the queries.
- The adversary finally outputs a guess d of the bit b.

We say the adversary *wins* if and only if $d = b$, and we define the adversary's advantage of winning this game against a protocol Π as:

$$\mathcal{A}^{\mathsf{c.unlink}}_{\Pi}(\mathcal{A}) := |\mathsf{Pr}[\mathcal{A} \text{ wins } \mathsf{Exp}^{\mathsf{c.unlink}}_{\mathcal{A}}(1^\lambda)] - \frac{1}{2}|.$$

Recalling the adversarial classes of [23,10], we call an adversary *narrow* if it may not use the Result oracle, permitting it to know whether the server has authenticated the client or not. The opposite of *narrow* are *wide* adversaries. Orthogoal to the use of the Result oracle, we also classify adversaries in terms of their use of the $\mathsf{Corrupt}$ oracle, which gives them access to the client's data. Thus, adversaries are *weak* if they cannot use the corruption oracle; they are *forward* if any corruption query may only be followed by more corruption queries[6]. Finally, adversaries are classified as *strong* if their access to oracles is unrestricted.

[5] In particular, we want to avoid trivial attacks, in which an adversary can distinguish a client simply because it is not in its original state (having already started the protocol run beforehand).

[6] In particular, the adversary may no longer free drawn clients, nor interact with servers or clients

We note that the 3GPP requirements outlined in Section 2.1 restrict their adversaries to "*eavesdroppers on the radio link*", which seems to indicate that they target *weak* adversaries that are either *narrow* or *wide*. Moreover, they restrict their adversaries to being passive only; in this paper, we also consider active weak attackers and thus obtain a better privacy guarantee.

Formalization. We quantify adversaries in terms of the following parameters: the adversarial class, which we abbreviate to α-c.unlink, with $\alpha \in \{\mathsf{nw}, \mathsf{ww}, \mathsf{nf}, \mathsf{wf}\}$ (for narrow- and wide-weak, and narrow-, respectively wide-forward adversaries); their execution time is t; the maximum number q_{exec} of sessions instantiated per client C; the maximum number q_{id} of user identification per session; and the maximum number q_G of queries to the function G. We formalize the following definitions.

Definition 1. *(Weak Unlinkability) A protocol Π is $(t, q_{\mathsf{exec}}, q_{\mathsf{id}}, q_G, \epsilon)$-nw/ww-client-unlinkable if no narrow/wide-weak-adversary running in time t, creating at most q_{exec} sessions and q_{id} user identification per session, and making at most q_G queries to the function G, has an advantage $\mathsf{Adv}_{\Pi}^{\mathsf{w.c.unlink}}(\mathcal{A}) \geq \epsilon$.*

Definition 2. *(Forward Unlinkability) A protocol Π is $(t, q_{\mathsf{exec}}, q_{\mathsf{id}}, q_G, \epsilon)$-nf/wf-client-unlinkable if no narrow/wide-forward-adversary running in time t, creating at most q_{exec} sessions and q_{id} user identification per session, and making at most q_G queries to the function G, has an advantage $\mathsf{Adv}_{\Pi}^{\mathsf{f.c.unlink}}(\mathcal{A}) \geq \epsilon$.*

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 and which "encompasses" all the cryptographic functions of the AKA protocol:

- $\mathsf{CreateCl}(\mathsf{Op}, \mathsf{LAI}) \rightarrow (\mathsf{C}_i, \mathsf{IMSI}, \mathsf{st}_{\mathsf{C}_i})$: this oracle creates a new, legitimate, free client, labelled C_i at a location LAI for which a server is already defined (else the oracle outputs $\perp$). The client's IMSI, its sequence number $\mathsf{Sqn}_{\mathsf{C}_i}$, and its secret key $\mathsf{sk}_{\mathsf{C}_i}$ are chosen uniformly at random from sets ID, ST, and $\mathcal{S}$ respectively; the past location and TMSI are set to a special symbol $\perp$. The client's operator key $\mathsf{sk}_{\mathsf{op}}$ is set to the key of the operator Op. The adversary is given the parameters $\mathsf{IMSI}, \mathsf{st}_{\mathsf{C}_i}$, and the label C_i (used later to select clients).
- $\mathsf{CreateS}(\mathsf{LAI}) \rightarrow \mathsf{S}_i$: this oracle generates a new server S_i at location LAI, if this location is not already defined for another server (else the oracle returns $\perp$).
- $\mathsf{Launch}(\mathsf{VC}, \mathsf{S}_j) \rightarrow (s, m)$: this oracle instantiates a new session (labelled by a unique identifier s) between the client associated with VC and the server S_j, and outputs an initial protocol message m from S_j to VC. This oracle keeps track of tuples $(s, \mathsf{VC}, \mathsf{S}_j)$.
- $\mathsf{DrawCl}(\mathsf{C}_i, \mathsf{C}_j) \rightarrow \mathsf{VC}$: on input a pair of client labels, this oracle generates a handle VC, which is a monotonic counter, if the following conditions are met: (a) both clients were free when the query was made; (b) both clients have the same current location value. If either condition is not met, the oracle outputs $\perp$. Else, depending on the value of the secret bit b, the challenger associates the handle VC either with C_i (if $b = 0$) or with C_j (if $b = 1$). The challenger stores the triple $(\mathsf{VC}, \mathsf{C}_i, \mathsf{C}_j)$ in a table $\mathcal{T}$.
- $\mathsf{FreeVC}(\mathsf{VC}) \rightarrow \perp$: on input the virtual handle VC, this oracle retrieves the values $\mathsf{C}_i, \mathsf{C}_j$ associated to VC in the table $\mathcal{T}$, aborting any ongoing protocol runs.
- $\mathsf{Relocate}(\mathsf{VC}, \mathsf{LAI}^*) \rightarrow \perp$: this oracle modifies the *current* location of the two clients $\mathsf{C}_i, \mathsf{C}_j$ associated with VC in $\mathcal{T}$, to LAI^*. In particular, the challenger does the following for each of the clients: (1) it sets $\mathsf{past.LAI} := \mathsf{curr.LAI}$; (2) it sets $\mathsf{curr.LAI} := \mathsf{LAI}^*$. Any protocol sessions still running for VC are aborted.
- $\mathsf{Send}(\mathsf{P}, s, m) \rightarrow m'$: for the first input, the adversary can input either a handle VC or a server identity S. In the former case, the oracle simulates sending the message m from the adversary to the client associated with VC in session s, returning either the party's message m' or $\perp$ if either s is not associated with VC or if VC does not exist. Parties may also return $m' = \perp$ as an error message. If the first input of this oracle is set to S, the oracle simulates sending the message from the adversary to the server S.
- $\mathsf{Execute}(\mathsf{VC}, \mathsf{S}, s) \rightarrow \tau$: this oracle simulates a complete protocol run between the client associated with VC and the server S. in the presence of a passive adversary. In particular, by alternating $\mathsf{SendToCl}$ and $\mathsf{SendToS}$ queries on genuinely-output messages, this oracle generates and outputs the transcript τ of the execution between the server S and the client C for which session s was created.

- $\mathsf{Result}(\mathsf{P}, s) \rightarrow \{0,1\}$: if $\mathsf{P} = \mathsf{VC}$, this oracle returns a bit indicating whether the client associated with VC has accepted the server that VC ran session s with. If $\mathsf{P} = \mathsf{S}$, then the bit indicates whether the server accepted the client. For the AKA protocol, an acceptance bit is equivalent to the confirmation of the key-exchange. If the session is incomplete or session s is not associated with P, the oracle returns $\perp$.
- $\mathsf{Corrupt}(\mathsf{C}) \rightarrow \{\mathsf{sk}, \mathsf{st}_\mathsf{C}, \mathsf{ID}_\mathsf{C}, (\mathsf{past.LAI}, \mathsf{curr.LAI})\}$: For a client C, this oracle returns the full state (static and ephemeral), the identifiers and the location information of client C.

Key-indistinguishability and Impersonation The notion of *key-indistinguishability* refers to the session keys computed as a result of authenticated key exchange, requiring that they be indistinguishable from random bitstrings of equal length. We use a subset of our previously-defined oracles, this time without anonymizing clients by use of handles. Additionally, the adversary may also know the ephemeral state (in our case, the sequence number) of both clients and servers. Since the state is updated in a probabilistic way, we give the adversary a mean 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 results in *adversarially controlled* parties.

In this model, participants (clients and servers) may run concurrent key-agreement executions; we denote the j-th execution of the protocol of party P as P_j. Each client C is associated with a *unique* identifier UID (which in the case of AKA is the IMSI); the identity of the server is of the form S_i. We simplify the key-indistinguishability model by abstracting location data (since it does not affect the security of the session keys).

We associate each instance P_i with a session ID sid, a partner ID pid, and an accept/reject bit accept. The partner ID pid is set to either the server S_i or to a user identifier UID; the session ID sid includes four values: the user ID given by client UID (and implicity sk_UID), the server identifier $\mathsf{ID}_{\mathsf{S}_\mathsf{i}}$, 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". 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. We allow adversaries to learn whether instances have terminated (by sending Send queries) and whether they have accepted or rejected their partners. 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.

Partners. Each instance of each party keeps track of a session ID string, denoted sid, consisting of the four values listed above. We define partners as party instances that share the same session ID. More formally:

Definition 3. ***[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.*

Formalization. In the key-indistinguishability game, we no longer need to formalize the drawing and freeing of clients; thus we do not use those oracles. We furthermore do not use the Relocation oracle. Instead, we give the adversary access to a Key-Reveal oracle, which returns session keys for an ongoing session terminated in an accepting state. The central oracle in this game is a real-or-random type of Testing oracle, which allow the adversary to know either a tuple of real session keys or a tuple or random keys of equal size. If the adversary can tell whether the keys are real or random, then she is said to win the key-indistinguishability game.

At the outset of the game, the challenger first generates the keys of all the n_{Op} operators and instantiates a number n_S of servers. Each server can then access synchronized copies of the n_{Op} operators internally. Each

copy of an operator Op run internally by a server S takes as input the identity of that specific server S (this models the fact that operators and servers communicate via an unspecified secure channel). The adversary is then allowed to query any of the oracles below. We implicitly assume that the $\mathsf{Test}^{\mathsf{K.Ind}}$ 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 $\mathsf{Test}^{\mathsf{K.Ind}}$ query as well.

Eventually, the adversary $\mathcal{A}$ outputs a bit d, which is a guess for the bit b used internally in the $\mathsf{Test}^{\mathsf{K.Ind}}$ oracle. The adversary *wins* if and only if: $b = d$ and $\mathcal{A}$ has queried a fresh instance to the $\mathsf{Test}^{\mathsf{K.Ind}}$ 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 4. ***[Freshness: key-indistinguishability.]*** *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) ***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:

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

Definition 5. ***[Key-indistinguishability.]*** *A key-agreement protocol* Π *has* $(t, q_{\mathsf{exec}}, q_{\mathsf{id}}, q_{\mathsf{serv}}, q_{\mathsf{Op}}, q_{\mathsf{G}}, \epsilon)$*-key-indistinguishability if no adversary running in time* t*, creating at most* q_{exec} *party instances, with* q_{id} *user identification per instance, corrupting at most* q_{serv} *servers, making at most* q_{Op} *OpAccess queries per operator per corrupted server, and making at most* q_{G} *queries to the function* G*, has an advantage* $\mathsf{Adv}^{\mathsf{K.Ind}}_{\Pi}(\mathcal{A}) \geq \epsilon$*.*

We also define the related notions of client- and server-impersonation security, which involves a slightly different notion of freshness.

Definition 6. ***[Freshness:*** $\mathsf{Imp.Sec}$***.]*** *An instance* P_i*, with session ID* sid *and partner ID* pid*, is* fresh *if: neither this instance nor a partner of* P_i *is adversarially-controlled; and if there exists no instance* P'_j *sharing session* sid *with the partner* $\mathsf{pid} = \mathsf{P}_i$ *(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* P_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* P'_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* P_i *via a* $\mathsf{Send}(m'')$ *query at time* $\mathsf{clock} = k'' > k' + 1$*.*

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

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

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

Definition 7. ***[Impersonation security.]*** *A key-agreement protocol Π is $(t, q_{\mathsf{exec}}, q_{\mathsf{id}}, q_{\mathsf{serv}}, q_{\mathsf{Op}}, q_{\mathsf{G}}, \epsilon)$-impersonation-secure if no adversary running in time t, creating at most q_{exec} party instances, with q_{id} user identification per instance, corrupting at most q_{serv} servers, making at most q_{Op} OpAccess queries per operator per corrupted server, and making at most q_{G} queries to the function G, has an advantage $\mathsf{Adv}^{\mathsf{C.Imp}}_{\Pi}(\mathcal{A}) \geq \epsilon$ or $\mathsf{Adv}^{\mathsf{S.Imp}}_{\Pi}(\mathcal{A}) \geq \epsilon$.*

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

Further security notions: Security w.r.t. servers In this section, we define two further notions, namely soundness and state-confidentiality with respect to servers. In these games the adversary is a malicious, but legitimate server S, and this is the only server we consider. We note that 3GPP specifications allow servers to communicate with each other, but *how* they do this is not apparent. We use the $\mathsf{OpAccess}$ oracle to give the server access to operators on specific clients, but change the syntax so that the oracle takes a single input, namely a client identifier C. We also demand that the output of this oracle represents material only for a single protocol session sid. The adversary uses the Send, $\mathsf{CreateCl}$, $\mathsf{NewInstance}$, $\mathsf{Execute}$, and $\mathsf{StReveal}$ oracles as described in the key-indistinguishability model. We additionally modify the corruption oracle, as noted below:

– $\mathsf{Corrupt}(\mathsf{P}) \rightarrow \mathsf{sk}_\mathsf{P}$: if P is a client, this oracle returns the long-term client key sk_P, but not sk_op (in this we keep faithful to the implementation of the protocol, which protects the key even from the user himself). If P is an operator, then this oracle returns sk_op and the list of tuples $(\mathsf{UID}, \mathsf{sk}_\mathsf{UID}, \mathsf{st}_{\mathsf{Op},\mathsf{C}})$ for all clients C subscribing with that operator.

State-Confidentiality. 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 8. *[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 9. ***[State-Confidentiality.]*** *A key-agreement protocol Π is $(t, q_\mathsf{exec}, q_\mathsf{id}, q_\mathsf{Op}, q_\mathsf{G}, \epsilon)$-state-confidential if no adversary running in time t, creating at most q_exec party instances, with at most q_id user identification per instance, making at most q_Op queries to any operator Op, and making at most q_G queries to the function 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 as in the impersonation game. The advantage of the adversary is defined as:

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

Definition 10. ***[Soundess w.r.t. server.]*** *A key-agreement protocol Π is $(t, q_\mathsf{exec}, q_\mathsf{id}, q_\mathsf{Op}, q_\mathsf{G}, \epsilon)$-server-sound if no adversary running in time t, creating at most q_exec protocol instances, with at most q_id user identification per instance, making at most q_Op queries to any operator Op, and making q_G queries to the function G, has an advantage $\mathsf{Adv}^{\mathsf{S.sound}}_{\Pi}(\mathcal{A}) \geq \epsilon$.*

3 The AKA protocol

3.1 Notations

Notations. Throughout the rest of the document, we denote the length of a bitstring x by $|x|$, while $\lfloor x \rfloor_{i..j}$ denotes bits i through j of x. If f is a function, then $y \leftarrow f(x)$ denotes that y was the output of f on input x.

Similarly, $y \xleftarrow{\$} \{0,1\}^n$ indicates that y is chosen uniformly at random from the set of bitstrings of length n, i.e. $\{0,1\}^n$. For strings x, y, the string $x\|y$ is the concatenation of x and y, while $x \oplus y$ denotes the exclusive OR (XOR) of x and y. For a bit b, $\mathbf{b}^n$ denotes the string $b\|b\|\ldots\|b$ of length n. We denote by $\perp$ the error message and by 1^λ the security parameter (in unary). The indistinguishability property is considered against the chosen plaintext attacks ($ind - cpa$). We denoted this property $\mathsf{ind} - \mathsf{cpa}$ to lighten the expressions.

3.2 Description of the AKA protocol

3G (and 4G) mobile networks use a variant of the AKA protocol, which is fully depicted in Appendix 3.2, in order to establish secure channels between mobile clients and servers. Ultimately, the server uses the secure channel to transmit a specific service to the mobile client.

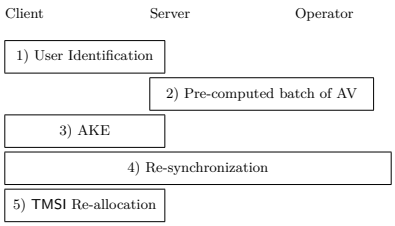


Fig. 2. The five phases of the AKA Procedure.

This protocol is actively run by clients and servers in the (selectively-active) presence of an operator. Servers and operators communicate over a secure and private channel; however, the server is considered only partially trusted. Section 2 describes in detail the setup of the three parties.

The AKA protocol consists of five phases. The first phase, *user identification*, is run by a client C and a server S on the *insecure channel* and it allows S to associate C to an IMSI value. A user ID request is first sent from the server to the client. The client's response is a tuple $(\mathsf{TMSI}, \mathsf{LAI})$, consisting of a temporary identifier and the local area identifier in which C received TMSI. If the LAI value corresponds to the LAI of S, then the latter searches for a tuple (TMSI, IMSI) in its own database; else S requests this tuple from the server S^* associated with LAI, over an unspecified channel. If no IMSI can be found, then the server demands the IMSI in clear. This procedure is the essential vulnerability leading to IMSI catcher attacks 3.3.

The second phase is run only optionally by the server S and the client's operator Op over a *secure channel*; its purpose is to enable S to then run a batch of AKE sessions with C. The server sends the client's IMSI to Op, which generates a batch of vectors AV each providing material for one out of a maximum of n sessions. For each vector, the operator's state with respect to the client $\mathsf{Sqn}_{\mathsf{S},\mathsf{C}}$ is augmented, and then the following values are generated: a fresh random value R; an server-authentication value Mac_S (for the values $\mathsf{Sqn}_{\mathsf{S},\mathsf{C}}$ and R); a client-authentication value Mac_C (for R only); the session keys CK and IK; and an anonymity key AK. Of these six values, the last five are computed by using each time a different cryptographic algorithm, denoted $\mathcal{F}_1, \ldots, \mathcal{F}_5$. In fact, the AKA protocol uses seven such algorithms, but two of them, denoted $\mathcal{F}_1^*, \mathcal{F}_5^*$, are only used in the *re-synchronization procedure*. The seven algorithms are generic, and can currently be instantiated in one of two ways, one using AES (called MILENAGE), the other using Keccak (called TUAK). Both $\mathcal{F}_1$ and $\mathcal{F}_1^*$ take as input the keys sk_C and $\mathsf{sk}_{\mathsf{op}}$, the random value R, and a sequence number $\mathsf{Sqn}_{\mathsf{S},\mathsf{C}}$. The other algorithms use the secret keys and the random value, but not the sequence number. At the end of this phase, the following values are sent to the server for each of the n sessions: $\mathsf{AV} = (\mathsf{R}, \mathsf{CK}, \mathsf{IK}, \mathsf{Mac}_\mathsf{S}, \mathsf{Mac}_\mathsf{C}, \mathsf{AMF}, \mathsf{AK} \oplus \mathsf{Sqn}_{\mathsf{S},\mathsf{C}})$, in which AMF is the Authentication Management Field is a 16-bit value used only in radio access specifications (for example E-UTRAN or non-3GPP access to EPS).

The sequence number $\mathsf{Sqn}_{\mathsf{S,C}}$ is notably *not* sent in clear to the server, but rather blinded by the value AK.

The third phase of the protocol, *authenticated-key-exchange*, is a mutual authentication and key agreement between the server and the client over the insecure channel. The server chooses the next vector available (if phase 2 was run, then this is the first tuple; else, for returning clients phase 3 is run directly, with the next authentication vector), and sends an *authentication challenge* consisting of the random value R and an authentication string $\mathsf{Autn} = (\mathsf{Sqn}_{\mathsf{S,C}} \oplus \mathsf{AK}) \| \mathsf{AMF} \| \mathsf{Mac}_{\mathsf{S}}$. The client uses R to compute AK, then it recovers $\mathsf{Sqn}_{\mathsf{S,C}}$ and verifies $\mathsf{Mac}_{\mathsf{S}}$. If the verification succeeds, and if the recovered Sqn is within a distance of Δ (a pre-defined constant) of the client's own state $\mathsf{Sqn}_{\mathsf{C}}$, then the client computes CK, IK, and the response $\mathsf{Mac}_{\mathsf{C}}$, sending this latter value to S; else, if the two sequence numbers are too far apart, then the client forces a *re-synchronization procedure*, which is the fourth phase of the protocol. If no re-synchronization is needed, then the client updates $\mathsf{Sqn}_{\mathsf{C}} := \mathsf{Sqn}_{\mathsf{Op,C}}$, and S verifies the received authentication value with respect to the $\mathsf{Mac}_{\mathsf{C}}$ sent by Op. If the verification succeeds, then the server sends an acknowledgement to Op and goes directly to phase five. If the verification fails, then the protocol is aborted.

The fourth phase of the protocol, *resynchronization*, is run by all three parties. The client essentially retraces the operator's steps, using its own sequence number $\mathsf{Sqn}_{\mathsf{C}}$ and computing the values $\mathsf{Mac}^*_{\mathsf{S}}$ and $\mathsf{AK}^* \oplus \mathsf{Sqn}_{\mathsf{C}}$ by using algorithms $\mathcal{F}_1^*$ and $\mathcal{F}_5^*$ (rather than $\mathcal{F}_1$ and $\mathcal{F}_5$), but keeping the same random value R. If the authentication string $\mathsf{Mac}^*_{\mathsf{S}}$ verifies for the Sqn value Op recovers, then Op resets its sequence number to $\mathsf{Sqn}_{\mathsf{C}}$ and sends to the server another batch of authentication sessions. The protocol restarts. We note that this phase is executed only optionally.

Finally, the fifth phase of AKA, TMSI *re-allocation*, is run by the server and client. As the first message of the record layer, the server sends an (unauthenticated) encryption of a new TMSI value to the client C, using the session key CK computed in phases 3 or 4. The encryption is done by means of the A5/3 algorithm detailed in TS 43.020 [4], run in cipher mode. The new TMSI value, called $\mathsf{TMSI}_{\mathsf{new}}$, is only permanently saved by the server if the client acknowledges the receipt; else, both values $\mathsf{TMSI}_{\mathsf{new}}$ and the old $\mathsf{TMSI}_{\mathsf{old}}$ are retained and can be used in the next authentication procedure. In appendix, the entire protocol is depicted in Figure 12.

3.3 Some Privacy breaches in AKA

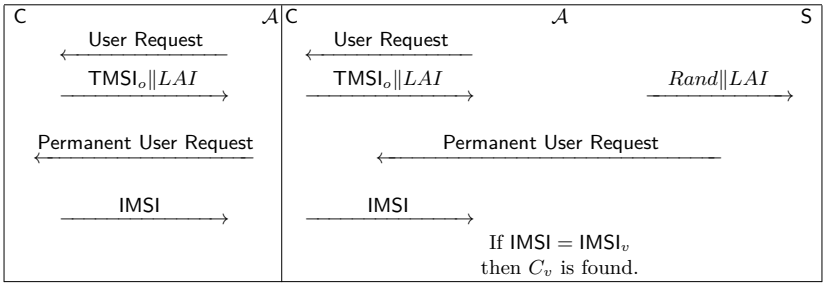


Fig. 3. Attacks based on TMSI.

To the best of our knowledge, the specifications do not detail how the permanent user identifiers are generated, we consider that all the user IDs are different.

We do not need to have a very formal/theoric analysis to point out the pure AKA protocol cannot guarantee the user unlinkability as defined previously. Indeed, the user identification based on temporary

identifiers independent to the permanent identifier is not sufficient. As related to our introduction, we clearly analyze the two main issues which restrict the AKA protocol to guarantee the user unlinkability. The first one is focused on the linkability of the failure messages. The next one is focused on the operational difficulties related to the TMSI. These weaknesses have been partially described by Arapinis and al. in [20,**?**].

Linkability of failure messages Arapinis and al [20] notably provides a novel practical attack establishing the traceability of a user based on the study of the failure messages. Considering a victim mobile client C_v, an adversary can detect its presence in a specific area, only considering the features of the failure messages, and replaying one old legitimate authentication vector including $(\mathsf{R}, \mathsf{Autn})$.

This latter can be replayed to any client each time it wants to check the presence or not of the victim client C_v. In fact this replay always implies a failure message. If this replayed authentication vector had been sent to C_v, the answer will be a "de-synchronization message". Indeed, the received MAC will be successfully verified, but the sequence number will be not in the correct range. Otherwise, the answer will be an "Mac failure message", because the the Mac could not be correct, except to a negligible probability of success $(1/2^{(128)})$ (due to the collision's probability). This difference can permit to trace a mobile client in a specific area just replaying a legitimate authentication vector. That represents a breach of the user's untraceability.

Operational difficulties with TMSI As we explained previously, 3GPP has standardized the use of temporary values TMSI instead of the basic permanent identifier IMSI. These temporary values are generated independently to the permanent identifier. At first glance, this option could guarantee at least the user identity confidentiality. However, the permanent identity IMSI is not as private as the sequence number. In fact, this value can be easily obtained by a weak active adversary. In the identification of the user, when the VLR cannot recognize the TMSI, the procedure is not aborted but the permanent identifier of the user is requested to the client. Thus, by a basic attack 3 an adversary can obtain the permanent identifier of a user during a fresh session. Obviously that represents a breach in the confidentiality of the user, that can also imply a breach in the intraceability. This weakness is exploited by the well-known "IMSI catcher" [6], which is the best known attack to mobile telephony users' privacy. This attack consists in forcing a mobile phone to reveal its permanent identity.

The Temporary identifier of each user must be used, as its name suggests, temporary. The specifications consider the uniqueness of the TMSI's use. As specified by the protocol, a TMSI re-allocation is provided. However, a basic active adversary can corrupt this uniqueness. Indeed, we note that the VLR does not de-allocate the old value TMSI_o without receiving the acknowledge message. So if an adversary drops this latter, the VLR has the both allocated temporary values: the old one TMSI_o and the new one TMSI_n.

For the next identification, the VLR will accept the both values to identify the user. So an adversary dropping the acknowledge message can easily force the replay of the same TMSI during different sessions. Moreover, we note that the new temporary identifier TMSI_o is only sent encrypted with the session key CK and without any integrity service. The mobile subscriber can not verify whether the received value is sent by the VLR or any malicious entity. Thus, any adversary can impose a false random value as new TMSI sending a random value instead of the ciphered temporary identifier.

As specified previously, each TMSI is associated to a specific location area denoting the VLR having generated this temporary identifier. The related LAI is sent in cleartext with the TMSI. Due to the previous detailed failure, an adversary can associated the IMSI with an LAI: that permits to know where the subscriber with the permanent identifier IMSI was located during the previous session.

Therefore, the use of temporary identifiers as specified by the specifications can not assure any privacy as specified.

We proceed to describe two of the more promising improvements to the AKA protocol, and show that these variants are vulnerable to client-unlinkability attacks.

The Arapinis variant. Arapinis et al. [20] propose an AKA variant which is supposed to ensure client unlinkability by avoiding failure-message-based linking, as described in the previous section. To avoid this attack, they propose to replace the both failure messages by two indistinguishable messages. The failure

message is now encrypted with a public key of the network and includes the IMSI, a constant Fail, a random value R and the current sequence number $\mathsf{Sqn_C}$. This latter is specially encrypted with an unlinkability key $\mathsf{UK} = f_{\mathsf{sk_C}}(\mathsf{R})$ in order to authenticate the error message. After receiving this generic error message, the network can deduce the cause of the failure from the IMSI and the sequence number, and sends the appropriate answer.

Moreover, they propose a fixed version of the identification protocol avoiding to expose the IMSI (basically the IMSI is sent in cleartext upon request by the network). It breaches both user identity confidentiality and untraceability. Thus, the IMSI is encrypted with an asymmetric-randomized-encryption (the same one as this one of the error messages) and they do not used any temporary value as usual. That permits to cancel the reallocation step. We suppose that if the VLR cannot recover the IMSI, the protocol is aborted (and they do not use a Permanent Identity Request). We note that this variant owns different practical issues. Indeed, authors proposes a simply representation of the network by concatening the VLR and HLR as an unique entity. For the security analysis, it is not an issue but it is the case for practical considerations. Contrary to their formal security proof, we consider that the fixed version from Arapinis does not assure the untraceability of the mobile subscriber. Indeed, we propose a new attack permitting to trace a victim client based on the knowledge of a user permanent identity. Its knowledge is reasonable since we can not consider this data as private. In a specific area, an adversary can trace a victim client C_v, which has its permanent identifier IMSI_v known by the adversary. This attack consists to replace the answer of the user identity request by a response with the permanent identifier of the victim client and analyze the behavior of the session. If the authentication data answer Res contains a failure message, the attacker considers the tested client as different from the victim client. Otherwise, the victim client has been found. This attack is detailed in the figure 4. We state the following result:

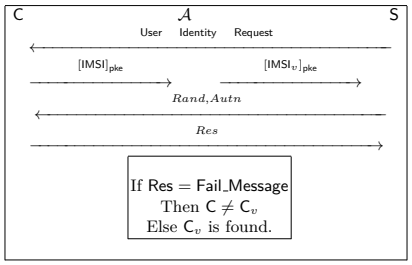


Fig. 4. Attack on the fixed version of Arapinis.

Lemma 1. *Let Π the protocol proposed by Arapinis and al. in [20]. Consider a $(t, 1, 1, 0)$-adversary $\mathcal{A}$ against the weak-privacy of the protocol Π running in time t, creating at most one party instance, running one user identification per each instance and making no extra query to the related internal cryptographic functions. The advantage of a such adversary is denoted $\mathsf{Adv}_{\Pi}^{\mathsf{ww\text{-}unlink}}(\mathcal{A})$. The protocol Π cannot guarantee the weak-privacy since there exists a such adversary $\mathcal{A}$ with a no-negligeable advantage.*

Proof. The attack we describe goes as follows:

1. At time $\mathsf{clock} = 0$, the challenger sets up the server.
2. At time $\mathsf{clock} = 1$, the adversary creates two clients C_0, C_1 from the oracles $\mathsf{CreateClient}(\mathsf{IMSI}^{\{0\}}, \mathsf{st}_0^{\{0\}})$ and $\mathsf{CreateClient}(\mathsf{IMSI}^{\{1\}}, \mathsf{st}_0^{\{1\}})$.
3. At time $\mathsf{clock} = 2 \rightarrow 3$, the adversary uses the oracle $\mathsf{DrawClient}(\mathsf{C}_0, \mathsf{C}_1)$ which returns at time $\mathsf{clock} = 3$ a virtual client $vc = \mathsf{C}_b$ following a chosen bit b.
4. At time $\mathsf{clock} = 4 \rightarrow 10$, the adversary runs an instance of the protocol Π between the virtual client and the terminal as follows: At time $\mathsf{clock} = 4$, the terminal sends a user identity request to the virtual

client; at time $\mathsf{clock} = 5$, the virtual client answers with the value UID such as: $\mathsf{UID} = [\mathsf{IMSI}^{\{b\}}]_{\mathsf{pke}_b}$; at time $\mathsf{clock} = 6$, the adversary intercepts this message (before arriving to the terminal) and forges a user identification $\mathsf{UID} = [\mathsf{IMSI}^{\{0\}}]_{\mathsf{pke}_0}$ for the client C_0 which will be sent to the terminal at time $\mathsf{clock} = 7$ (instead of the user identifier computed by the virtual client); at time $\mathsf{clock} = 8$, the terminal receives the related user identifier and generates an authentication challenge which is sent to the virtual client vc at time $\mathsf{clock} = 9$; this latter sends its authentication answer Res at time $\mathsf{clock} = 10$ which is eavesdropped by the adversary.

5. At time $\mathsf{clock} = 11$, since the eavesdropped authentication answer Res, the adversary tries to guess the bit b as follows: if the message Res contains a failure message, then the adversary considers the virtual client as the client C_0 (that implies $b' = 0$). Otherwise, it considers the virtual client as the client C_1 (that implies $b' = 1$).
6. At time $\mathsf{clock} = 12$, the adversary returns its guess b'.

This attack is detailed in the figure 4. It is clear that the advantage of a such adversary is 1. Thus, the protocol cannot guarantee the weak-privacy. We note that in practice if the adversary $\mathcal{A}$ cannot detect the de-synchronization, some "false positive" can be appear: indeed, during the tested session, if the victim client is de-synchronized, the adversary could not trace it. But the frequency of a such event (i.e this one of a synchronization procedure) is very weak. Moreover, we can use this attack procedure at least twice to reduce the probability to obtain a false positive.

The van den Broek variant. Van den Broek et al. [7] recently proposed an IMSI catcher countermeasure; in this improved variant, avoid sending the IMSI in clear by replacing (IMSI, TMSI) tuples by an upgradeable pseudonym denoted PMSI. Their modified identification phase is exclusively done by means of these pseudonyms. The PMSI is chosen by the operator and sent with the authentication challenge in the preparation phase, encrypted together with the sequence number with a new secret key that is assumed to be shared by clients and their operators. The ciphertext is used as the random value R in the authentication challenge. Indeed, a successful session of the AKA protocol, ending in the establishment of new session keys, can only be attained if the PMSI is correctly updated. This variant is described in detail in [7].

From a practical point of view, using the operator at each key-exchange session is costly, and something that the original AKA design tries to avoid. Furthermore, though this variant successfully prevents IMSI catchers, it does not address client unlinkability. The pseudonym PMSI can be intercepted in one session; if this session is then aborted, the PMSI can be replayed in a second session, thus leading to user linkability. Furthermore, the protocol is vulnerable to the attack based on linking failure messages, as presented by Arapinis et al. Thus, if Π denotes the protocol proposed by van den Broek et al., it holds that:

Lemma 2. *There exists a $(t, 2, 1, 0)$-adversary $\mathcal{A}$ against the narrow-weak-client-unlinkability of Π running in time t, initiating two protocol sessions, and making no query to the internal cryptographic function G, which has an advantage $\mathsf{Adv}_{\Pi}^{\mathsf{ww\text{-}unlink}}(\mathcal{A}) = \frac{1}{2}$ (and a probability of 1) to win the game.*

Proof. The attack we describe goes as the Arapinis attack based on the linkability of failure messages:

1. At time $\mathsf{clock} = 0$, the challenger sets up the server.
2. At time $\mathsf{clock} = 1$, the adversary creates two clients C_0, C_1 from the oracles $\mathsf{CreateClient}(\mathsf{IMSI}^{\{0\}}, \mathsf{st}_0^{\{0\}})$ and $\mathsf{CreateClient}(\mathsf{IMSI}^{\{1\}}, \mathsf{st}_0^{\{1\}})$.
3. At time $\mathsf{clock} = 2 \rightarrow 3$, the adversary uses the oracle $\mathsf{DrawClient}(\mathsf{C}_0, \mathsf{C}_1)$ which returns at time $\mathsf{clock} = 3$ a virtual client $vc = \mathsf{C}_b$ following a chosen bit b.
4. At time $\mathsf{clock} = 4 \rightarrow 5$, the adversary runs an instance of the protocol Π between the virtual client and the terminal via the oracle $\mathsf{Execute}(\mathsf{vc}, \mathsf{s})$ which returns the transcript of an instance s at time $\mathsf{clock} = 5$. This transcript includes the authentication challenge denoted $(\mathsf{R}, \mathsf{Autn})$.
5. At time $\mathsf{clock} = 6 \rightarrow 12$, the adversary runs an instance $s+1$ of the protocol Π between the virtual client and the terminal as follows: at time $\mathsf{clock} = 6$, the terminal sends a user identity request to the virtual

client; at time $\mathsf{clock} = 7$, the virtual client sends the user identifier UID to the terminal; at time $\mathsf{clock} = 8$, the terminal receives the related user identifier and generates an authentication challenge which is sent to the virtual client vc; at time $\mathsf{clock} = 9$, the adversary intercepts the fresh authentication challenge (before arriving to the virtual client) and sends the previous authentication challenge $(\mathsf{R}, \mathsf{Autn})$ to the virtual client at $\mathsf{clock} = 10$ (obtains in a previous session on the client $\mathsf{C}_0 := \mathsf{DrawClient}(\mathsf{C}_0, \mathsf{C}_0)$); at time $\mathsf{clock} = 11$, the virtual client sends its authentication answer Res at $\mathsf{clock} = 12$ which is eavesdropped by the adversary.

6. At time $\mathsf{clock} = 13$, since the eavesdropped authentication answer Res, the adversary tries to guess the bit b as follows: if the message Res contains a "de-synchronization message", then the adversary considers the virtual client as the client C_0 (that implies $b' = 0$). Otherwise, it considers the virtual client as the client C_1 (that implies $b' = 1$).
7. At time $\mathsf{clock} = 14$, the adversary returns its guess b'.

4 Our proposal: PrivAKA

In this section, we propose a new fixed variant of the AKA protocol which respects the weak-unlinkability and key-indistinguishability as defined in our privacy model 2. Obviously, it also guarantees the client-, server-impersonation. Moreover, our fixed variant takes into account practical requirements of the VLR in addition to provide the security properties. The VLR is an interoperable server in a cellular network that supports roaming functions for subscribers outside the coverage area of their own HLR. The VLR needs to manage the generation of the temporary identifier to avoid collision between subscriber from different operators. Indeed, if the HLR generates the new temporary identifiers, it cannot avoid collision between identifiers of user from different HLR but only avoids collision between its own subscribers. Then, the VLR cannot store any secret data shared with any user due to its interoperability feature, except the storage of the relation between the permanent and temporary identifiers for each subscriber. Indeed, the VLR communicates with user from any operator. So, this feature implies that the secret data also needs to be inter-operable, i.e a same secret data common for all the operators. We note that our fixed protocol does not increase the number of exchanged between the USIM and ME and respect the 3GPP's desire to limit communication with the HLR. Moreover, we are the first to seriously consider such practical requirements.

4.1 Description of our variant

Instead of five phases, our variant only consists of three. Our protocol is designed to not require re-synchronization (which was phase 4 in AKA), and we include the TMSI reallocation (which was phase 5 in AKA) as part of the key-exchange phase (phase 3). In our construction, we use a public-key encryption scheme PKE= (PKE.KGen,PKE.Enc,PKE.Dec), such that each operator has a certified public and secret key-pair denoted as $(\mathsf{pke}_{\mathsf{Op}}, \mathsf{ske}_{\mathsf{Op}})$. We assume that the client stores only its own operator's public key (and its certificate) internally. In particular, we do not give encryption keys to the servers in order to minimize key-management issues. We also use a secure authenticated encryption scheme $\mathsf{AE} = (\mathsf{AE.KGen}, \mathsf{AE.Enc}, \mathsf{AE.Dec})$. Though these can be instantiated generically, we use **AES-GCM** [5] for the AE scheme and (EC)IES [25] for the PKE scheme. We depict our variant in Figure 5, and indicate in the grey boxes the differences to the classical AKA procedure. Just as the original protocol, our variant starts with an *identification* phase, run by the client and the server over the insecure channel. The server sends an identification request which includes a random value that we denote R_{id}. The client forges an user identification answer following a flag $\mathsf{flag}_{\mathsf{TMSI}}$ managed by the client: it uses either a pre-exchanged fresh temporary identifier TMSI (if the previous protocol has been accepted and if the user stayed in the same area: $\mathsf{flag}_{\mathsf{TMSI}} = 1$) or it sends a public-key encryption of the concatenation of the received random value R_{id} and the evaluation of the function $\mathcal{F}_5$ on input the client secret key, the operator's secret key, the random value R_{id} and the client's IMSI (only if it does not own a fresh temporary identifier or if the user is located in a new area: $\mathsf{flag}_{\mathsf{TMSI}} = 0$). We demand

that the output size of the PKE scheme for this input size is always equal to the length of the TMSI. A such flag $\mathsf{flag_{TMSI}}$ is only used to restrict the computation of an encrypted permanent identity and not for security reasons. In both cases, the client also appends the identity of the operator Op it subscribes to, to the message. Moreover, we assume that the client can detect when it moves in a new area. In this case, it allocates the flag at one.

Intuitively, if the client stayed in the same area ($\mathsf{flag_{TMSI}} = 0$), then the TMSI the client stored came from the server it is currently communicating with, hence the server can find the $(\mathsf{TMSI}, \mathsf{IMSI})$ association in its database. Otherwise, the client does not use its TMSI value, but rather encrypts a function of the IMSI with the operator's public key. The function is symmetric-key, requiring knowledge of the client- and operator keys, and it is not replayable due to the fresh identification randomness $\mathsf{R_{id}}$. Upon receiving a string of the form (m, Op), the server first checks whether the message m is a TMSI present in its database; if so, it retrieves the IMSI to which this value corresponds; else, it assumes that m is a ciphertext, and it sends it to the operator Op for decryption.

Phase 2, *preparation*, is run over a secure channel, between the server and the operator. If the server received a valid TMSI in the previous phase, the preparation phase being with the server sending the corresponding IMSI to the operator; else, the server forwards the received ciphertext and the associated random value $\mathsf{R_{id}}$. The operator proceeds similarly to the standard AKA preparation procedure, with the following differences:

- We add as input to each cryptographic function a server-specific value $\mathsf{Res_S}$ for a server with identity $\mathsf{S} = \mathsf{VLR}$. This is to prevent attacks in which the adversary replays authentication vectors from one network to another, as presented by Zhang [17]. We also use the constant AMF which is sent in clear, as an additional input.
- We add the sequence number $\mathsf{Sqn_{Op,C}}$ to each of the cryptographic functions apart from $\mathcal{F}_5$. Since the sequence number is an ephemeral value, which is updated, this guarantees freshness even if the randomness R is repeated.
- We introduce an index value $\mathsf{idx_{Op,C}}$ which essentially prevents the repetition of a challenge using the same sequence number. This value is essential in preventing a desynchronization of sequence number values. We note that the client also keeps track of a similar index $\mathsf{idx_C}$, which will play a role in the key-exchange phase, as detailed below.

In the final phase, *authenticated-key-exchange*, which is run between the client and the server, the server sends a random value R, the authentication string Autn, an authenticated encryption of the new TMSI, using the keys $(\mathsf{CK}, \mathsf{IK})$ derived for this session. The client proceeds similarly to the original AKA procedure, recovering AK by using the random value R, then checking $\mathsf{Mac_S}$. If the authentication cannot be verified, the procedure is aborted, and the new TMSI is disregarded. Else, the user computes CK, IK and decrypts the received authenticated encryption string to find the TMSI value and the operator's index $\mathsf{idx_{Op,C}}$. Then, C checks the freshness of the sequence number, i.e. it verifies if one of the following two conditions is correct:

- $\mathsf{Sqn_C} = \mathsf{Sqn}^{\{i\}}$.
- $\mathsf{Sqn_C} = \mathsf{inc}(\mathsf{Sqn}^{\{i\}})$ and $\mathsf{idx_{Op,C}} = \mathsf{idx_C} + 1$,

If the protocol is run normally, the first of these conditions is the one that will hold. However, if the previous session is aborted after receiving the server's authentication challenge, then the two sequence numbers may become desynchronized by one step (the second condition). Further desynchronization is prevented by the use of an index, which indicate whether the authentication string for a particular $\mathsf{Sqn_{Op,C}}$ has already been used or not. If the first condition holds, then the client's internal index is reset; else, the index is incremented by 1. The client updates the sequence number only upon successful authentication. If none of these conditions are verified, the procedure is aborted and does not use of a resynchronization procedure.

Finally, the user computes a response $\mathsf{Res} := \mathcal{F}_2(\mathsf{keys}, \mathsf{R}^{\{i\}}, \mathsf{Sqn}^{\{i\}}, \mathsf{Res_S}, \mathsf{AMF})$, sends Res, then stores the TMSI and the new index value. The server checks Res against the prepared value $\mathsf{Mac_C}$ (else, if no response is received, the procedure is aborted).

One notable exception to the original AKA protocol is that whenever an abort occurs on the server's side, the second phase – *preparation* – is used instead of simply querying the next vector in the prepared batch. Though this might seem more inefficient, we note that an abort only occurs in the presence of an adversary, which is considered to be a rare event. We detail the procedure upon aborts in Figure 6.

Internal cryptographic algorithms:

In our variant, we have modified the inputs of the internal cryptographic algorithms, notably to include the sequence number and the new value $\mathsf{Res_S}$. Thus, we need to provide an update of these algorithms to consider these modifications. As specified in specifications, the AKA protocol can be based on the different sets of algorithms: TUAK and MILENAGE.

To preserve backwards compatibility, we propose to keep and update these both sets. Moreover, our variant requires an algorithm of authenticated encryption. We require to use the well-know standard **AES-GCM** [5]. It is denoted AE.

Considering our modifications, we do not use the function $\mathcal{F}_1^*$ and $\mathcal{F}_5^*$ since we have dropped the resynchronization.

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

- keys: the couple of the both 128-bit (or 256-bit) keys: the subscriber key sk and the operator key $\mathsf{sk_{op}}$.
- Sqn (except for the function $\mathcal{F}_5$): a 48-bit sequence number.
- AMF (except for the function $\mathcal{F}_5$): a 16-bit authentication field management.
- R: a 128-bit random value.
- $\mathsf{Res_S}$: a 128-bit (public?) value characterizing the visited network.

We note that the function $\mathcal{F}_5$ behaves differently because they do not consider the sequence number in inputs (contrary to the currently version, where $\mathcal{F}_1$ and $\mathcal{F}_1^*$ behave differently).

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

To assure a stronger 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 the both values Top_C and Temp as follows:

$$\mathsf{Top}_C = \mathsf{sk_{op}} \oplus \mathsf{AES_{sk_c}}(\mathsf{sk_{op}}),$$

$$\mathsf{Temp} = \mathsf{AES_{sk_c}}(\mathsf{R} \oplus \mathsf{Top}_C \oplus \mathsf{Res_S}).$$

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

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

with the five integers $\mathsf{r_1} = 0$, $\mathsf{r_2} = 24$, $\mathsf{r_3} = 48$, $\mathsf{r_4} = 64$ and $\mathsf{r_5} = 96$ 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 $\mathsf{c_5}[124] = 1$.

This is also described in the Figure 7.

Update of the TUAK algorithms: TUAK is an alternative set of MILENAGE based on the internal permutation of the Keccak [8]. 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.

As the no-updated TUAK algorithms, we compute firstly the value Top_C as follows:

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

We note that the values AN, Inst', Inst, Pad are the same as used in the no-updated TUAK algorithms and Key the (padded) subscriber key.

At this point, the behavior of the functions $\mathcal{F}_5$ diverges from that of the other functions. To generate the related output, we compute the value Val_1 and for the others ones, we compute the value Val_2 which differ as follows:

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

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

- **Output $\mathcal{F}_1$:** $\mathsf{Mac_S} = \lfloor \mathsf{Val}_2 \rfloor_{0..127}$,
- **Output $\mathcal{F}_2$:** $\mathsf{Mac_C} = \lfloor \mathsf{Val}_2 \rfloor_{256..383}$,
- **Output $\mathcal{F}_3$:** $\mathsf{CK} = \lfloor \mathsf{Val}_2 \rfloor_{512..639}$,
- **Output $\mathcal{F}_4$:** $\mathsf{IK} = \lfloor \mathsf{Val}_2 \rfloor_{768..895}$,
- **Output $\mathcal{F}_5$:** $\mathsf{AK} = \lfloor \mathsf{Val}_1 \rfloor_{0..47}$.

This is also depicted in Figure 8.

We note that the multi-output property is, as the no-updated 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).

4.2 Privacy and Security Analysis

Weak Client Unlinkability of our fixed variant In this section, we prove that our fixed variant of the AKA protocol achieves weak client unlinkability if we assume that the internal functions assures indistinguishability, pseudorandomness and unforgeability properties. The weak-client-unlinkability resistance of our fixed variant is proved as follows:

Theorem 1. [ww-unlink − Resistance.]
Let $G_{:}\{0,1\}^{\kappa} * \{0,1\}^{d} * \{0,1\}^{t} * \{0,1\}^{t} \to \{0,1\}^{n}$ be our unitary function described in section 4.2 and Π our fixed variant of the AKA *protocol specified in section 4. Consider a $(t, q_{\mathsf{exec}}, q_{\mathsf{id}}, q_G, q_{\mathsf{AE}}, q_{\mathsf{PKE}})$-adversary $\mathcal{A}$ against the weak privacy* ww-unlink*-security of the protocol Π running in time t, creating at most q_{exec} party instance, with at most q_{id} user identification per instance and making at most q_G, q_{AE}, q_{PKE} queries to respectively the functions G,* AE *and* PKE*. The advantage of this adversary is denoted $\mathsf{Adv}^{\mathsf{ww\text{-}unlink}}_{\Pi}(\mathcal{A})$. Then, there exist $(t' \sim O(t), q' = q_{\mathsf{exec}} + q_G)$-adversary $\mathcal{A}_1$ and $(t' \sim O(t), q' = 2 \cdot q_{\mathsf{exec}} + q_G)$-$\mathcal{A}_2$ against respectively the unforgeability and pseudorandomness of the function G, an $(t'' \sim O(t), q'' = q_{\mathsf{exec}} + q_{\mathsf{AE}})$-adversary $\mathcal{A}_3$ against the ae-security of the function* AE*, $(t''' \sim O(t), q''' = q_{\mathsf{exec}} \cdot q_{\mathsf{id}} + q_{\mathsf{PKE}})$-adversaries $\mathcal{A}_4$ against the indistinguishability of the function* PKE*, $(t, q_{\mathsf{exec}}, q_{\mathsf{id}}, 0, 0, q_G, q_{\mathsf{AE}}, q_{\mathsf{PKE}})$-adversary $\mathcal{A}_5$ against the key-secrecy of the protocol Π such that:*

$$\mathsf{Adv}^{\mathsf{ww\text{-}unlink}}_{\Pi}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{K.Ind}}_{\Pi}(\mathcal{A}_5) + \frac{1 + (q_{\mathsf{exec}} \cdot q_{\mathsf{id}})^2}{2^{|\mathsf{TMSI}|}} + \frac{q_{\mathsf{exec}}^2}{2^{|\mathsf{R}|}} + \frac{(q_{\mathsf{exec}} \cdot q_{\mathsf{id}})^2}{2^{|\mathsf{R_{id}}|}} + n_{\mathsf{C}} \cdot (3 \cdot \mathsf{Adv}^{\mathsf{mac}}_{G}(\mathcal{A}_1) + \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}_2) + 4 \cdot \mathsf{Adv}^{\mathsf{ae}}_{\mathsf{AE}}(\mathcal{A}_3) + \mathsf{Adv}^{\mathsf{ind-cca2}}_{\mathsf{PKE}}(\mathcal{A}_4)).$$

Proof. Our proof has the following hops:

Game $\mathbb{G}_0$: This game works as the ww-unlink-game stipulated in section 2.

Game $\mathbb{G}_1$: We modify the original game $\mathbb{G}_0$ to ensure that the random values $\mathsf{R_{id}}$ and R used by honest server instances are always unique. The related security loss due to the collisions between each respective random in two different instances is given by the following expression:

$$\left| \Pr[\mathcal{A}_{\mathbb{G}_0}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins}] \right| \leq \frac{q_{\mathsf{exec}}^2}{2^{|\mathsf{R}|}} + \frac{(q_{\mathsf{exec}} \cdot q_{\mathsf{id}})^2}{2^{|\mathsf{R_{id}}|}}.$$

Game $\mathbb{G}_2$: The game $\mathbb{G}_2$ behaves as the game $\mathbb{G}_1$ with the restriction consisting to abort the protocol if one of the three following events happen:

- **Event 1:** The adversary $\mathcal{A}$ has forged the user identification answer.
- **Event 2:** The adversary $\mathcal{A}$ has forged the authentication challenge.
- **Event 3:** The adversary $\mathcal{A}$ has forged the authentication response.

Briefly, we modify the original game $\mathbb{G}_1$ to ensure that the adversary cannot forge these three messages. Indeed, a classic attack permits to recover the user identity of the client from the ability to forge any of these three messages. A such adversary behaves as follows: the adversary chooses one of the both clients (C_0 or C_1) related to the virtual client VC and forges a message x related to the current transcript. During the session, it drops the related message from the honest server and sends the forged message. If with a such modification the session is accepted, then the chosen client is the client to this current session. Otherwise, the other client is the good one.

Now, focus on each event. Firstly, if the adversary wants to forge a fresh user identification response ($\mathsf{event\ 1}$), she has two options: either guess the fresh temporary identifier TMSI or an encrypted version of the permanent identifier related to the random value $\mathsf{R_{id}}$ included in the user identification request. To guess the fresh temporary value, either it guess the fresh one from the old previous ones or it decrypts the authenticated encryption of the fresh one from the previous session. The TMSIś are independently chosen, so they are no way to guess the fresh TMSI from the old one, except randomly, or replaying one of these previous TMSI. The probability of a such success is at least $(1 + (q_{\mathsf{exec}} \cdot q_{\mathsf{id}})^2)/2^{|\mathsf{TMSI}|}$. The ability to recover the fresh TMSI from the encrypted version is restricted by the security of the authenticated encryption and the key-indistinguishability of the session keys. Indeed, the fresh TMSI is sent authenticated and encrypted by the algorithm AE, which is based on session keys and not long-term keys. Moreover, we note that the fresh TMSI is encrypted with the index which is predictable. So, if an adversary can forge a correct output of the authenticated encryption AE (i.e the related input includes a correct index), it can impose its own temporary value and uses it for the next user identification. So the probability of a such recovery is at most $\mathsf{Adv}_{\Pi}^{\mathsf{K.Ind}}(\mathcal{A}) + n_{\mathsf{C}} \cdot (2 \cdot \mathsf{Adv}_{\mathsf{AE}}^{\mathsf{ae}}(\mathcal{A}))$. Then, the ability to forge a correct encrypted permanent identity is restricted by the ability to forge a correct output of the function G with the IMSI and without the private key. The success probability of a such ability is $n_{\mathsf{C}} \cdot \mathsf{Adv}_{G}^{\mathsf{mac}}(\mathcal{A})$. So the security loss due to the ability to forge an fresh identification response is $\frac{1+(q_{\mathsf{exec}} \cdot q_{\mathsf{id}})^2}{2^{|\mathsf{TMSI}|}} + \mathsf{Adv}_{\Pi}^{\mathsf{K.Ind}}(\mathcal{A}) + n_{\mathsf{C}} \cdot (2 \cdot \mathsf{Adv}_{\mathsf{AE}}^{\mathsf{ae}}(\mathcal{A}) + \mathsf{Adv}_{G}^{\mathsf{mac}}(\mathcal{A}))$.

Now, focus on the second event involving to forge an authentication challenge. We recall that a such challenge is split in four parts: the random value R, a masked value of the current sequence number $val = \mathsf{AK} \oplus \mathsf{Sqn}$, a message authentication code mac generated by the function G which takes in inputs the private keys keys, the random value R and the sequence number Sqn, and the couple of the next temporary identifier and the current index encrypted by an authenticated encryption AE. In a fresh instance, we have two options to forge an authentication challenge: either the adversary guesses a fresh authentication challenge based on the current sequence number or it replays an old challenge based on a previous used sequence number. For the first option, the complexity to forge a such challenge is restricted by the unforgeability of the function G. Indeed even if the adversary knows the fresh sequence number, which is in practice masked by one-time-pad, it cannot forge a fresh message authentication code without the related message authentication code. Moreover, we note that the index is only implied for the second condition of the freshness verification and the new TMSI will be only used for the next session. With the second condition, the best option for the adversary consists to replay the three first parts of the previous authentication challenge included in an aborted session and

tries to forge an authenticated and encrypted version of fresh index and fresh TMSI. Indeed, when a protocol is aborted after that the server has sent the authentication challenge, the next authentication challenge is based on the same sequence number. To forge a fresh authenticated and encrypted value, the adversary chooses any non-nil value for the index value. We note that the adversary needs to know it because a such challenge including the index idx is considered as fresh only if there are exactly idx previous sessions aborted by a drop of the related authentication response. Moreover, we do not need to the chosen temporary value. The ability to forge a such challenge is restricted by the security of the chosen algorithm of authenticated encryption. So, considering both conditions, the success probability is at most $n_{\mathsf{C}} \cdot (\mathsf{Adv}^{\mathsf{mac}}_{G}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{ae}}_{\mathsf{AE}}(\mathcal{A}))$.

Finally, we focus on the third event involving to forge an authentication response. This value is only composed by the value Res, which is an output of the function G. So the success probability is at most $n_c \cdot \mathsf{Adv}^{\mathsf{mac}}_{G}(\mathcal{A})$. Thus, we obtain

$$\begin{aligned}\big|\Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}]\big| \leq \frac{1+(q_{\mathsf{exec}}\cdot q_{\mathsf{id}})^2}{2^{|\mathsf{TMSI}|}}\\ +\mathsf{Adv}^{\mathsf{K.Ind}}_{\Pi}(\mathcal{A}) + 3\cdot n_{\mathsf{C}}\cdot(\mathsf{Adv}^{\mathsf{ae}}_{\mathsf{AE}}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{mac}}_{G}(\mathcal{A})).\end{aligned}$$

Game $\mathbb{G}_3$: We modify the game $\mathbb{G}_3$ to replace outputs to call to the functions by truly randoms, i.e consistent values which are independent of the input, but the same input gives same output. We argue that the security loss is precisely the advantage of the adversary against the pseudorandomness of the internal cryptographic functions and the related security of PKEand AE. It holds that:

$$\begin{aligned}\big|\Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}]\big| \leq n_{\mathsf{C}}\cdot(\mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A})\\ +\mathsf{Adv}^{\mathsf{ind-cca2}}_{\mathsf{PKE}}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{ae}}_{\mathsf{AE}}(\mathcal{A})).\end{aligned}$$

Winning the game $\mathbb{G}_3$: At this point, the adversary plays a game which consider to recover the bit b with a whole-randomized protocol. Assume that the adversary cannot learn any information about the related client in a such transcript. Thus, the adversary has only one option: guess the bit b without any specific information. So we obtain the following probability of winning the game $\mathbb{G}_3$:

$\Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}] = \frac{1}{2}$.

Security Statement This yields the following result:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ww\text{-}unlink}}_{\Pi}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{K.Ind}}_{\Pi}(\mathcal{A}_5) + \frac{1+(q_{\mathsf{exec}}\cdot q_{\mathsf{id}})^2}{2^{|\mathsf{TMSI}|}} +\\ \frac{q^2_{\mathsf{exec}}}{2^{|\mathsf{R}|}} + \frac{(q_{\mathsf{exec}}\cdot q_{\mathsf{id}})^2}{2^{|\mathsf{R_{id}}|}} + n_{\mathsf{C}}\cdot(3\cdot\mathsf{Adv}^{\mathsf{mac}}_{G}(\mathcal{A}_1) +\\ \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}_2) + 4\cdot\mathsf{Adv}^{\mathsf{ae}}_{\mathsf{AE}}(\mathcal{A}_3) + \mathsf{Adv}^{\mathsf{ind-cca2}}_{\mathsf{PKE}}(\mathcal{A}_4)).\end{aligned}$$

Key-Indistinguishability of our fixed variant We consider the key-indistinguishability property, denoted K.Ind, as the guarantee that the session keys of honest sessions are indistinguishable from random. In the model previously detailed in section 2.2, we consider the session ID sid of each instance as follows: UID, $\mathsf{ID}_{\mathsf{S_i}}$, R, $\mathsf{R_{id}}$, $\mathsf{idx_C}$ and the value $\mathsf{Sqn_C}$, that are agreed upon the session. As stipulated we can prove the key secrecy of our fixed variant of the AKA protocol, under indistuiguishability, unforgeability and pseudorandomness properties of the different internal functions. This property is defined as follows:

Theorem 2. [K.Ind – **resistance**.] *Let $G : \{0,1\}^{\kappa} \times \{0,1\}^{d} \times \{0,1\}^{t} \times \{0,1\}^{t} \to \{0,1\}^{n}$ be our specified function specified in section 4.2 and Π our fixed variant of the* AKA *protocol specified in section 4.2. Consider a $(t, q_{\mathsf{exec}}, q_{\mathsf{id}}, q_{\mathsf{serv}}, q_{\mathsf{Op}}, q_G, q_{\mathsf{AE}}, q_{\mathsf{PKE}})$-adversary $\mathcal{A}$ against the* K.Ind*-security of the protocol Π, running in time t and creating at most q_{exec} party instance, with at most q_{id} user identification per instance, corrupting at most q_{serv} servers, making at most q_{Op} OpAccess queries per operator per corrupted server, and making*

at most q_G, q_{AE}, q_{PKE} *queries to respectively the functions* G, AE *and* PKE. *Denote the advantage of this adversary as* $\mathsf{Adv}^{\mathsf{K.Ind}}_{\Pi}(\mathcal{A})$. *Then there exist a* $(t' \approx O(t), q' = q_G + q_{\mathsf{exec}})$*-mac-adversary* $\mathcal{A}_1$ *on* G, *a* $(t' = O(t), q' = q_G + 2 \cdot q_{\mathsf{exec}} + 5 \cdot q_{\mathsf{serv}} \cdot q_{\mathsf{Op}})$*-prf-adversary* $\mathcal{A}_2$ *on* G, *a* $(t' = O(t), q' = q_G + q_{\mathsf{exec}})$*-ind-cpa-adversary* $\mathcal{A}_3$ *on* G,*a* $(t' = O(t), q' = q_{\mathsf{exec}} + q_{\mathsf{AE}})$*-ae-adversary* $\mathcal{A}_4$ *on* AE, *and a* $(t' = O(t), q' = q_{\mathsf{exec}} \cdot q_{\mathsf{id}} + q_{\mathsf{PKE}})$*-ind-cca2-adversary* $\mathcal{A}_5$ *on* PKE *such that:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{K.Ind}}_{\Pi}(\mathcal{A}) \leq n_{\mathsf{C}} \cdot (\frac{(q_{\mathsf{exec}} \cdot q_{\mathsf{id}})^2}{2^{|\mathsf{R}_{\mathsf{id}}|}} + \frac{(q_{\mathsf{exec}} + q_{\mathsf{serv}} \cdot q_{\mathsf{Op}})^2}{2^{|\mathsf{R}|}} + \\ \mathsf{Adv}^{\mathsf{mac}}_{G}(\mathcal{A}_1) + \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}_2) + \mathsf{Adv}^{\mathsf{ind-cpa}}_{G}(\mathcal{A}_3) \\ + \mathsf{Adv}^{\mathsf{ae}}_{\mathsf{AE}}(\mathcal{A}_4) + \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{PKE}}(\mathcal{A}_5)).\end{aligned}$$

Proof. Our proof has the following hops.

Game $\mathbb{G}_0$: This game works as the $\mathsf{K.Ind}$-game stipulated in our security model 2.2, but including the new oracles. 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 both games have 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 outputs $\perp$). The output of the oracle depends on the value $\mathtt{type} \in \{\mathtt{sub}, \mathtt{op}, \mathtt{all}\}$. 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 $\mathtt{type} = \mathtt{all}$, we return the both values sk_{P}, $\mathsf{sk}_{\mathsf{op}}$. If $\mathtt{type} \in \{\mathtt{sub}, \mathtt{all}\}$, 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).

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

Game $\mathbb{G}_2$: We modify $\mathbb{G}_1$ to only allow interactions with a single client (any future UReg 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 proceed as follows: for any adversary $\mathcal{A}_{\mathbb{G}_1}$ winning the game $\mathbb{G}_1$ with a no-negligible success probability $\epsilon_{\mathbb{G}_1}$, we propose to construct a generic 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 UReg query that $\mathcal{A}_{\mathbb{G}_1}$ makes to its own UReg 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 $\mathsf{rsk}_{\mathsf{Op}^*}$ (for all operator Op^* such that $\mathsf{Op}^* \neq \mathsf{Op}$), 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 UReg 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 two types of clients: the *brothers* of the target client (i.e the clients which have the same operator as the target client) and the others ones. For these latter, denoted 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 $\frac{1}{2}$. 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:

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

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 and R_{id} in two different instances) of

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

Game $\mathbb{G}_4$: This game behaves as the game $\mathbb{G}_3$ with the restriction to only interact with only one server (any future UReg calls for a server would be answered with an error symbol $\perp$). The benefices lost is the ability to obtain some authentication vectors from corrupted servers. Such authentication vectors can either give information about the used sequence number and the long term keys or forge a fresh challenge replaying some parts of these vectors. We recall that the challenge is split in four 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), a mac computed with the function G and an authenticated and encrypted version of the next temporary identifier and the current index. Moreover, we note that all the call of the function G take in input a specific value of the related server, denoted $\mathsf{ID}_{\mathsf{S}_i}$. The corrupted servers permit to obtain vectors based on the fresh sequence number but different random and different server identifier. 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. 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

$$\big|\Pr[\mathcal{A}_{\mathbb{G}_4}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}]\big| \leq \mathsf{Adv}_G^{\mathsf{ind-cpa}}(\mathcal{A}) + \mathsf{Adv}_G^{\mathsf{mac}}(\mathcal{A}).$$

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). We

argue that the security loss is precisely the advantage of the adversary $\mathcal{A}$ against the pseudorandomness of function G, and the security of respectively PKE and AE. Note that the total number of queries to the related functions are at most $2 \cdot q_G$ and one q_{AE} and one q_{PKE} per honest instance(thus totaling at most $\mathsf{q}_G + (2 \cdot q_{\mathsf{exec}})$ queries to the function G, q_{exec} queries to the function PKE and q_{exec} queries to the function AE).

$$\begin{aligned}\big|\Pr[\mathcal{A}_{\mathbb{G}_4}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_5}\mathsf{wins}]\big| \leq \mathsf{Adv}^{\mathsf{ind-cca2}}_{\mathsf{PKE}}(\mathcal{A}) + \\ \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{ae}}_{\mathsf{AE}}(\mathcal{A}).\end{aligned}$$

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:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{K.Ind}}_{\Pi}(\mathcal{A}_{\mathbb{G}_0}) \leq n_{\mathsf{C}} \cdot (\frac{(q_{\mathsf{exec}} \cdot q_{\mathsf{id}})^2}{2^{|\mathsf{R}_{\mathsf{id}}|}} + \frac{(q_{\mathsf{exec}} + q_{\mathsf{serv}} \cdot q_{\mathsf{Op}})^2}{2^{|\mathsf{R}|}} \\ + \mathsf{Adv}^{\mathsf{mac}}_{G}(\mathcal{A}_1) + \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}_2) + \mathsf{Adv}^{\mathsf{ind-cpa}}_{G}(\mathcal{A}_3) \\ + \mathsf{Adv}^{\mathsf{ae}}_{\mathsf{AE}}(\mathcal{A}_4) + \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{PKE}}(\mathcal{A}_5)).\end{aligned}$$

This concludes the proof.

Impersonation of our fixed variant

Proof. We present first the client-impersonation resistance proof, then the equivalent statement for server impersonations.

C.Imp-resistance

Game $\mathbb{G}_0$: This game works as the $\mathsf{C.Imp}$-game: When $\mathcal{A}$ stops, she wins if there exists an instance S_i that ends in an accepting state with session ID sid and partner ID pid such that: (a) pid is not adversarially controlled ($\mathsf{sk}_{\mathsf{pid}}$ has not been corrupted), (b) no other instance C_i exists for $\mathsf{pid} = \mathsf{S}_i$ that ends in an accepting state, with session ID sid.

Game $\mathbb{G}_1$: We modify the game to allow the new $\mathsf{Corrupt}(\mathsf{P}, \mathtt{type})$ query from the previous proof. 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, as in the previous proof, giving a security loss of:

$$\Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}] \geq \frac{1}{n_{\mathsf{C}}} \Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins}] + \frac{1}{2} \cdot \big(1 - \frac{1}{n_{\mathsf{C}}}\big)$$

Game $\mathbb{G}_3$: We now restrict the adversary to using a single server As detailed in the key-indistinguishability proof, the related security loss is given by: This give us a security loss

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

Game $\mathbb{G}_4$: We modify $\mathbb{G}_4$ to replace outputs to calls to all the internal cryptographic functions by truly random, but consistent values, and as before, we lose a term:

$$\begin{aligned}\left|\Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_4}\mathsf{wins}]\right| \leq \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{ae}}_{\mathsf{AE}}(\mathcal{A}) \\ + \mathsf{Adv}^{\mathsf{ind-cca2}}_{\mathsf{PKE}}(\mathcal{A}).\end{aligned}$$

Game $\mathbb{G}_5$: We modify $\mathbb{G}_4$ to ensure that the random values sampled by honest server instances are always unique. As in the unlinkability proof, this yields a loss of:

$$\left|\Pr[\mathcal{A}_{\mathbb{G}_4}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_5}\mathsf{wins}]\right| \leq \frac{(q_{\mathsf{exec}} \cdot q_{\mathsf{id}})^2}{2^{|\mathsf{R}_{\mathsf{id}}|}} + \frac{q_{\mathsf{exec}}^2}{2^{|\mathsf{R}|}}.$$

Winning $\mathbb{G}_5$: At this point, the adversary plays a game with a single client and server. A server instance S_i only accepts $\mathcal{A}_{\mathbb{G}_5}$, if the latter can generate a fresh identification response ID and 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 values Res and ID computed by C_i are 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 sk_C is unknown; (b) guessing the key 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 $2^{-|\mathsf{sk}_\mathsf{C}|}$. The third option occurs with a probability of $2^{-|\mathsf{Res}|} + 2^{-|\mathsf{ID}|}$ per session, thus a total of $q_{\mathsf{exec}} \cdot (2^{-|\mathsf{Res}|} + q_{\mathsf{id}} \cdot 2^{-|\mathsf{ID}|})$. Thus,

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

Security statement: This yields the following result:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{C.Imp}}_{\Pi}(\mathcal{A}_{\mathbb{G}_0}) \leq n_\mathsf{C} \cdot (\mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}_1) + \mathsf{Adv}^{\mathsf{ae}}_{\mathsf{AE}}(\mathcal{A}_2) \\ + \mathsf{Adv}^{\mathsf{ind-cca2}}_{\mathsf{PKE}}(\mathcal{A}_3) + \frac{(q_{\mathsf{exec}} \cdot q_{\mathsf{id}})^2}{2^{|\mathsf{R}_{\mathsf{id}}|}} \\ + \frac{(q_{\mathsf{exec}})^2}{2^{|\mathsf{R}|}} + \frac{q_{\mathsf{exec}}}{2^{|\mathsf{Res}|}} + \frac{1}{2^\kappa} + \frac{q_{\mathsf{exec}} \cdot q_{\mathsf{id}}}{2^{|\mathsf{ID}|}}).\end{aligned}$$

S.Imp−resistance

Game $\mathbb{G}_0$: This game works as the S.Imp-game. The adversary $\mathcal{A}$ wins if there exists an instance C_i that ends in an accepting state with session ID sid and partner ID and pid s.t.: (a) $\mathsf{pid} = \mathsf{S}$, (b) no instance S_j exists such as S_j and C_i has the same session ID sid, (c) C_i and these partners are not adversarially controlled.
Game $\mathbb{G}_1$: We add the new query $\mathsf{Corrupt}(\mathsf{P}, \mathtt{type})$ as in the previous proof, such 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, as in the previous proofs, and lose:

$$\Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}] \geq \frac{1}{n_\mathsf{C}} \Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins}] + \frac{1}{2} \cdot \left(1 - \frac{1}{n_\mathsf{C}}\right)$$

Game $\mathbb{G}_3$: Again, we restrict the adversary to only one server. This give us a security loss

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

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 and as before, it holds that:

$$\left|\Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_4}\mathsf{wins}]\right| \leq \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{ae}}_{\mathsf{AE}}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{ind-cca2}}_{\mathsf{PKE}}(\mathcal{A}).$$

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 C_i of the target client, for some target session sid. Note that the $\mathsf{Mac_S}$ value computed (for verification) 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 R; 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 session ID. 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, thus a total of $q_{\mathsf{exec}} \cdot 2^{-|\mathsf{Mac_S}|}$. Thus,

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

Security statement: This yields the following result:

$$\mathsf{Adv}^{\mathsf{S.Imp}}_{\Pi}(\mathcal{A}_{\mathbb{G}_0}) \leq n_{\mathsf{C}} \cdot \big(\frac{q_{\mathsf{exec}}}{2^{|\mathsf{Mac_S}|}} + \frac{1}{2^{\kappa}} + \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}_1) + \mathsf{Adv}^{\mathsf{ae}}_{\mathsf{AE}}(\mathcal{A}_2) + \mathsf{Adv}^{\mathsf{ind-cca2}}_{\mathsf{PKE}}(\mathcal{A}_3)\big).$$

Soundness and key-confidentiality of our fixed variant.

Theorem 3. [$\mathsf{St.conf}-$**resistance**.] *Let G and G^* be our specified functions specified in section 4.2 and Π our fixed variant of the* AKA *protocol specified in section 4.2. Consider a $(t, q_{\mathsf{exec}}, q_{\mathsf{id}}, q_{\mathsf{Op}}, q_G, q_{G^*}, q_{\mathsf{AE}}, q_{\mathsf{PKE}})$-adversary $\mathcal{A}$ against the* $\mathsf{St.conf}$*-security of the protocol Π, running in time t and executing q_{exec} sessions of the protocol Π, making at most q_{Op} queries to any operator, and making q_G, q_{G^*}, $q_{\mathsf{AE}}, q_{\mathsf{PKE}}$ queries to respectively the functions G, G^*,* AE *and* PKE*resp. q_{G^*}) queries to the function G (resp. G^*). Denote the advantage of this adversary as $\mathsf{Adv}^{\mathsf{St.conf}}_{\Pi}(\mathcal{A})$. Then there exist a $(t' \approx O(t), q' = q_G + 5 \cdot q_{\mathsf{Op}} + 2 \cdot q_{\mathsf{exec}})$-prf-adversary $\mathcal{A}_1$ on G, a $(t' = O(t), q' = q_{\mathsf{exec}} + q_{\mathsf{AE}})$-ae-adversary $\mathcal{A}_2$ on* AE*, a $(t' = O(t), q' = q_{\mathsf{exec}} \cdot q_{\mathsf{id}} + q_{\mathsf{PKE}})$-ind-cca2-adversary $\mathcal{A}_3$ on* PKE*, and $(t' \approx O(t), q' = q_{G^*})$-prf-adversary $\mathcal{A}_4$ on G^* a such that:*

$$\mathsf{Adv}^{\mathsf{St.conf}}_{\Pi}(\mathcal{A}) \leq n_{\mathsf{C}} \cdot (\frac{1}{2^{|\mathsf{sk_C}|}} + \frac{1}{2^{|\mathsf{sk_{Op}}|}} + \frac{1}{2^{|\mathsf{Sqn}|}} + \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}_1) + \mathsf{Adv}^{\mathsf{ae}}_{\mathsf{AE}}(\mathcal{A}_2) + \mathsf{Adv}^{\mathsf{ind-cca2}}_{\mathsf{PKE}}(\mathcal{A}_3)) + \mathsf{Adv}^{\mathsf{prf}}_{G^*}(\mathcal{A}_4).$$

Proof. 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 $\epsilon_{\mathbb{G}_0}$, we propose to construct a generic 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 $\mathsf{rsk}_{\mathsf{Op}^*}$. 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 $\mathsf{sk}_{\mathsf{C}^*}$ and a sequence number $\mathsf{Sqn}_{\mathsf{C}^*}$, returning the latter value. Moreover, each new client registered with the operator Op (resp. any Op^*) is associated with the related operator key $\mathsf{sk}_{\mathsf{op}}$(resp. $\mathsf{rsk}_{\mathsf{Op}^*}$).

We distinguish two types of clients: the *brothers* of the target client (i.e the clients which have the same operator as the target client) and the others ones. For these latter, denoted 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 $\mathsf{rsk}_{\mathsf{Op}^*}$.

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 $\mathsf{sk}_{\mathsf{C}^*}$, $\mathsf{rsk}_{\mathsf{Op}^*}$, and the current value of $\mathsf{Sqn}_{\mathsf{C}^*}$. 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 $\frac{1}{2^{|\mathsf{sk_C}|}} + \frac{1}{2^{|\mathsf{sk_{op}}|}} + \frac{1}{2^{|\mathsf{Sqn_C}|}}$. 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:

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

Note that adversary $\mathcal{A}_{\mathbb{G}_1}$ makes no query with respect to $\mathcal{A}_{\mathbb{G}_0}$.

Game $\mathbb{G}_2$: We modify $\mathbb{G}_1$ to only allow interactions with a single client (any future $\mathsf{CreateCl}(\mathsf{Op})$ 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 $\mathsf{sk}_{\mathsf{op}}$. We proceed as follows: for any adversary $\mathcal{A}_{\mathbb{G}_1}$ winning the game $\mathbb{G}_1$ with a no-negligible success probability $\epsilon_{\mathbb{G}_1}$, we propose to construct a generic 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 $\mathsf{sk}_{\mathsf{C}^*}$ and $\mathsf{Sqn}_{\mathsf{C}^*}$, 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 $\mathsf{sk}_{\mathsf{C}^*}$ and the current value of $\mathsf{Sqn}_{\mathsf{C}^*}$. Indeed, since the adversary can corrupt any operator key, she requires the oracle of G^* permitting to simulate all the queries of the brothers of the target client.

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 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:

$$\Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}] \geq \frac{1}{\mathsf{n}_{\mathsf{C},\mathsf{Op}}} \Pr[\mathcal{A}_{\mathbb{G}_1}\mathsf{wins}] + \frac{1}{2} \cdot \left(1 - \frac{1}{\mathsf{n}_{\mathsf{C},\mathsf{Op}}}\right)$$

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

Note that adversary $\mathcal{A}_{\mathbb{G}_2}$ makes no extra query with respect to $\mathcal{A}_{\mathbb{G}_1}$.

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^*, and related security of the functions PKE and AE.

$$\begin{aligned}\left|\Pr[\mathcal{A}_{\mathbb{G}_3}\mathsf{wins}] - \Pr[\mathcal{A}_{\mathbb{G}_2}\mathsf{wins}]\right| \leq\ & \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{prf}}_{G^*}(\mathcal{A}) \\ & + \mathsf{Adv}^{\mathsf{ae}}_{\mathsf{AE}}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{ind-cca2}}_{\mathsf{PKE}}(\mathcal{A}).\end{aligned}$$

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:

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

Security statement: This yields the following result:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{St.conf}}_{\Pi}(\mathcal{A}_{\mathbb{G}_0}) \leq n_{\mathsf{C}} \cdot \Big(& \frac{1}{2^{|\mathsf{sk}_{\mathsf{C}}|}} + \frac{1}{2^{|\mathsf{sk}_{\mathsf{Op}}|}} + \frac{2}{2^{|\mathsf{Sqn}|}} \\ & + \mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}_1) + \mathsf{Adv}^{\mathsf{ae}}_{\mathsf{AE}}(\mathcal{A}_2) \\ & + \mathsf{Adv}^{\mathsf{ind-cca2}}_{\mathsf{PKE}}(\mathcal{A}_3)\Big) + \mathsf{Adv}^{\mathsf{prf}}_{G^*}(\mathcal{A}_4).\end{aligned}$$

We provide the following theorem about the client-impersonation, denoted S.sound-security, of the fixed variant of the AKA protocol.

Theorem 4. [$\mathsf{S.sound} - $ **resistance**.] *Let* $G : \{0,1\}^{\kappa} \times \{0,1\}^{d} \times \{0,1\}^{t} \times \{0,1\}^{t} \to \{0,1\}^{n}$ *be our specified function specified in section 4.2 and* Π *our fixed variant of the* AKA *protocol specified in section 4.2. Consider a* $(t, q_{\mathsf{exec}}, q_{\mathsf{Op}}, q_G, q_{\mathsf{AE}}, q_{\mathsf{PKE}})$*-adversary* $\mathcal{A}$ *against the* S.sound*-security of the protocol* Π*, running in time* t *and executing* q_{exec} *sessions of the protocol, making at most* q_{Op} *queries to any operator, and making* q_G, $q_{\mathsf{AE}}, q_{\mathsf{PKE}}$ *queries to respectively the functions* G, AE *and* PKE*. Denote the advantage of this adversary as* $\mathsf{Adv}^{\mathsf{S.sound}}_{\Pi}(\mathcal{A})$*. Then there exist a* $(t' \approx O(t), q' = q_G + q_{\mathsf{exec}})$*-mac-adversary* $\mathcal{A}_1$ *on* G*,* $(t' \approx O(t), q' = q_G + 2 \cdot q_{\mathsf{exec}} + 5 \cdot q_{\mathsf{Op}})$*-prf-adversary* $\mathcal{A}_2$ *on* G*, a* $(t' = O(t), q' = q_G + q_{\mathsf{exec}})$*-ind-cpa-adversary* adv_3 *on* G*, a* $(t' = O(t), q' = q_{\mathsf{exec}} + q_{\mathsf{AE}})$*-ae-adversary* adv_4 *on* AE *and a* $(t' = O(t), q' = q_{\mathsf{exec}} + q_{\mathsf{PKE}})$*-ind-cca2-adversary* $\mathcal{A}_5$ *on* PKE *such that:*

$$\mathsf{Adv}_{\Pi}^{\mathsf{S.sound}}(\mathcal{A}) \leq n_{\mathsf{C}} \cdot (\frac{q_{\mathsf{exec}}}{2^{|\mathsf{Mac_S}|}} + \frac{1}{2^{\kappa}} + \mathsf{Adv}_{G}^{\mathsf{mac}}(\mathcal{A}_1) + \mathsf{Adv}_{G}^{\mathsf{prf}}(\mathcal{A}_2) + \mathsf{Adv}_{G}^{\mathsf{ind-cpa}}(\mathcal{A}_3) + \mathsf{Adv}_{\mathsf{AE}}^{\mathsf{ae}}(\mathcal{A}_4) + \mathsf{Adv}_{\mathsf{PKE}}^{\mathsf{ind-cca2}}(\mathcal{A}_5)).$$

Proof. **Game** $\mathbb{G}_0$: This game works as the game S.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 S.sound-game, we have a *legitimate-but-malicious* adversary.

Game $\mathbb{G}_1$: We consider the game $\mathbb{G}_1$ as the 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 as follows: for any adversary $\mathcal{A}_{\mathbb{G}_0}$ winning the game $\mathbb{G}_0$ with a no-negligible success probability $\epsilon_{\mathbb{G}_0}$, we propose to construct a generic adversary $\mathcal{A}_{\mathbb{G}_1}$ winning the game $\mathbb{G}_1$ with a black-box access to the adversary $\mathcal{A}_{\mathbb{G}_0}$.

Both adversaries play her related game with oracles. The following oracles are similar in the two games: Send, CreateCl, NewInstance, 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 $\mathsf{AV} = (\mathsf{R}, \mathsf{Autn}, \mathsf{Mac_C}, \mathsf{CK}, \mathsf{IK})$. Simulating the answer of the oracle $\mathsf{OpAccess}(\mathsf{C}_i)$, the $\mathcal{A}_{\mathbb{G}_1}$ uses the query $\mathsf{Execute}(S, \mathsf{C}_i)$ (with the server related to the *legitimate-but-malicious* adversary) and $\mathsf{Reveal}(\mathsf{C}, i)$.

Now, focus on the simulation of the Corrupt answer. We recall that we have two possible inputs: a client or an operator. In the Corrupt oracle takes in input a client, the adversary $\mathcal{A}_{\mathbb{G}_1}$ uses its own Corrupt oracle to obtain the related answer. If the input is an operator, $\mathcal{A}_{\mathbb{G}_1}$ needs to forge the following values: the operator key $\mathsf{sk_{op}}$, 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 Cof this operator.

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

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

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

$$\mathsf{Adv}_{\Pi}^{\mathsf{S.sound}}(\mathcal{A}_{\mathbb{G}_0}) \leq n_{\mathsf{C}} \cdot (\frac{q_{\mathsf{exec}}}{2^{|\mathsf{Mac_S}|}} + \frac{1}{2^{\kappa}} + \mathsf{Adv}_{G}^{\mathsf{mac}}(\mathcal{A}_1) + \mathsf{Adv}_{G}^{\mathsf{prf}}(\mathcal{A}_2) + \mathsf{Adv}_{G}^{\mathsf{ind-cpa}}(\mathcal{A}_3) + \mathsf{Adv}_{\mathsf{AE}}^{\mathsf{ae}}(\mathcal{A}_4) + \mathsf{Adv}_{\mathsf{PKE}}^{\mathsf{ind-cca2}}(\mathcal{A}_5)).$$

Updated TUAK algorithms security In order to prove the pseudorandomess, unforgeability and indistinguishability of our updated TUAK algorithms, we assume that the truncated keyed internal Keccak permutation is a good pseudorandom function. We propose two generic constructions to model the updated TUAK algorithms: a first one, denoted G_{tuak} when the secret is based on the subscriber key sk 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 $\mathsf{f}_{\mathsf{Keccak}}$.

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

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

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

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} acts as a generalization of the specific TUAK algorithms:

$$\begin{aligned}
\mathcal{F}_1(\mathsf{sk_{op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Sqn}, \mathsf{Res_S}, \mathsf{AMF}) &= G_{\mathsf{tuak}}(\mathsf{sk_C}, \mathsf{inp}_1, 0, 127) \\
&= G^*_{\mathsf{tuak}}(\mathsf{sk_{op}}, \mathsf{inp}^*_1, 0, 127), \\
\mathcal{F}_2(\mathsf{sk_{op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Sqn}, \mathsf{Res_S}, \mathsf{AMF}) &= G_{\mathsf{tuak}}(\mathsf{sk_C}, \mathsf{inp}_1, 256, 383) \\
&= G^*_{\mathsf{tuak}}(\mathsf{sk_{op}}, \mathsf{inp}^*_1, 256, 383), \\
\mathcal{F}_3(\mathsf{sk_{op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Sqn}, \mathsf{Res_S}, \mathsf{AMF}) &= G_{\mathsf{tuak}}(\mathsf{sk_C}, \mathsf{inp}_1, 512, 639) \\
&= G^*_{\mathsf{tuak}}(\mathsf{sk_{op}}, \mathsf{inp}^*_1, 512, 639), \\
\mathcal{F}_4(\mathsf{sk_{op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Sqn}, \mathsf{Res_S}, \mathsf{AMF}) &= G_{\mathsf{tuak}}(\mathsf{sk_C}, \mathsf{inp}_1, 768, 895) \\
&= G^*_{\mathsf{tuak}}(\mathsf{sk_{op}}, \mathsf{inp}^*_1, 768, 895), \\
\mathcal{F}_5(\mathsf{sk_{op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Res_S}) &= G_{\mathsf{tuak}}(\mathsf{sk_C}, \mathsf{inp}_2, 0, 47) \\
&= G^*_{\mathsf{tuak}}(\mathsf{sk_C}, \mathsf{inp}^*_2, 0, 47),
\end{aligned}$$

with:

$\mathsf{inp}_1 = \mathsf{sk_{op}}\|\mathsf{cst}_1\|\mathsf{cst}_3$,, $\mathsf{inp}_2 = \mathsf{sk_{op}}\|\mathsf{cst}_1\|\mathsf{cst}_3$,
$\mathsf{inp}^*_1 = \mathsf{cst}_1\|\mathsf{keys}\|\mathsf{cst}_3$,, $\mathsf{inp}^*_2 = \mathsf{cst}_1\|\mathsf{keys}\|\mathsf{cst}_3$,
$\mathsf{cst}_1 = \mathsf{Inst}\|\mathsf{AN}\|0^{192}\|(\mathsf{Inst}'\|\mathsf{AN}\|\mathsf{R}\|\mathsf{AMF}\|\mathsf{Sqn})$,
$\mathsf{cst}_3 = \mathsf{Res_S}\|\mathsf{Pad}\|1\|0^{192}$,

We define the cascade constructions G_{tuak} and G^*_{tuak} based on the function f and f^* as follows:

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

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 $0 \leq i \leq j \leq 1600$.

We express the required security properties of the generalization G_{tuak} (resp. G^*_{tuak}) under the prf-security of the function f (resp. f^*). Since the construction of the two functions, while we cannot prove the latter property, we can conjecture that the advantage of a prf-adversary would be of the form:

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

for any adversary $\mathcal{A}$ running in time t and making at most q queries at its challenger. Here, m is the output's size of our function f and T_f is the time to do one f computation on the fixed RAM model of computation and c_1 and c_2 are two constants depending only on this model. In other words, we assume that the best attacks are either a exhaustive key search or a specific attack on this construction. This attack uses the fact that the permutation is public and can be easily inverted. Even if the protocol truncates the permutation, if the output values are large, and an exhaustive search on the missing bits is performed, it is possible to

invert the permutation and recover the inputs. Since the secret keys is one of the inputs as well as some known values are also inputs, it is then possible to determine which guesses of the exhaustive search are correct guess or incorrect ones. Finally, if the known inputs are shorter than the truncation, false positives can happen due to collisions and we have to filter the bad guesses. However, if the number of queries is large enough, it is possible to filter these bad guesses and uniquely recover the keys.

Pseudorandomness and Unforgeability of TUAK algorithms. We begin by reducing the prf-security of G_{tuak} to the prf-security of the function f. This implies the mac-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 5. [$\mathsf{prf}-$**security for** $\mathbf{G}^*_{\mathsf{tuak}}$.] *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:*

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

Proof. We construct the adversary $\mathcal{A}_{f^*}$ using a prf-adversary $\mathcal{A}_{G^*}$. The latter uses $\mathcal{A}_{f^*}$ as a challenger for a prf-game $\mathbb{G}_{\mathsf{prf}(f^*)}$ and can only communicate with $\mathcal{A}_{f^*}$ whereas $\mathcal{A}_{f^*}$ has access to a challenger for f^*. To begin with, the challenger $\mathcal{C}^{\mathsf{prf}}_{f^*}$ chooses a bit b and a private $\mathsf{sk_{op}} \in \{0,1\}^\kappa$. If $b = 0$, it assigns f^* to a random function and if $b = 1$, it assigns f^* to the specific internal function.

The adversary $\mathcal{A}_{f^*}$ waits for queries from $\mathcal{A}_{G^*}$ of the form (m, a, b), with $m = m^{(1)}\|m^{(2)}\|m^{(3)} \in \{0,1\}^d$, and $a, b \in \{0,1\}^t$ and responds as follows:

- It queries its challenger $\mathcal{C}^{\mathsf{prf}}_{f^*}$ for inputs $(m^{(1)}\|m^{(3)}, 0, 256)$ and receives the value Out_1.
- Then, it computes $\mathsf{Out}_2 = f^*(\mathsf{Out}_1, m^{(2)}\|m^{(3)}, a, b)$.
- It returns the value Out_2.

We note that the two first bullets permits to generate $G^*(\mathsf{sk_{op}}, m, a, b) = \mathsf{Out}_2$. This step is repeated up to a total of q queries from $\mathcal{A}_{g^*}$, with a and b fixed.

At some point, $\mathcal{A}_{g^*}$ halts and outputs a guess d of the bit b. The prf-adversary $\mathcal{A}_{f^*}$ chooses its guess b' as $b' = d$ and forwards it to $\mathcal{C}^{\mathsf{prf}}_{f^*}$, which verifies if $b = b'$.

We analyze this simulation. Recall that the challenger responded either with a random value (if its internal bit b was set to 0) or with the output of the function $f^*(\mathsf{sk_{op}}, m^{(1)}\|m^{(3)}, 0, 256)$ (if its internal bit was set as 1).

Thus, the output Out_2 matches either the output of a random function or the output of the function $G^*(\mathsf{sk}, m, a, b)$. So the prf-adversary $\mathcal{A}_{f^*}$ simulates perfectly a prf-challenger of G. Thus, we have:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{prf}}_{f^*}(\mathcal{A}_{f^*}) &= \big|\Pr[\mathcal{A}_{f^*} \to 1 \mid b = 1] - \Pr[\mathcal{A}_{f^*} \to 1 \mid b = 0]\big| \\
&= \big|\Pr[b = b' | b = 1] - \Pr[b = b' | b = 0]\big| \\
&= \big|\Pr[b = d | b = 1] - \Pr[b = d | b = 0]\big| \\
&= \big|\Pr[d' = d | b = 1] - \Pr[d' = d | b = 0]\big| \\
&= \big|\Pr[\mathcal{A}_{G^*} \to 1 \mid b = 1] - \Pr[\mathcal{A}_{G^*} \to 1 \mid b = 0]\big| \\
&= \big|\mathsf{Adv}^{\mathsf{prf}}_{G^*}(\mathcal{A}_{G^*})\big|.
\end{aligned}$$

Theorem 6. [$\mathsf{prf}-$**security for** $\mathbf{G}_{\mathsf{tuak}}$.] *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*, 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* $(t' \approx 2 \cdot t, q' = 2 \cdot q)$*-adversary* $\mathcal{A}$*' with an advantage* $\mathsf{Adv}^{\mathsf{prf}}_{f}(\mathcal{A}')$ *of winning against the pseudorandomness of* f *such that:*

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

Proof. We construct the adversary $\mathcal{A}_f$ using a prf-adversary $\mathcal{A}_G$ The latter uses $\mathcal{A}_f$ as a challenger for a prf-game $\mathbb{G}_{\mathsf{prf}(f)}$ and can only communicate with $\mathcal{A}_f$ whereas $\mathcal{A}_f$ has access to a challenger for f. To begin with, the challenger $\mathcal{C}^{\mathsf{prf}}_f$ chooses a bit b and a private $\mathsf{sk} \in \{0,1\}^\kappa$. If $b = 0$, it assigns f to a random function and if $b = 1$, it assigns f to the specific internal function.

The adversary $\mathcal{A}_f$ waits for queries from $\mathcal{A}_G$ of the form (m, a, b), with $m = m^{(1)} \| m^{(2)} \| m^{(3)} \in \{0,1\}^d$, and $a, b \in \{0,1\}^t$ and responds as follows:

- It queries its challenger $\mathcal{C}^{\mathsf{prf}}_f$ for inputs $(m^{(1)} \| m^{(3)}, 0, 256)$ and receives the value Out_1.
- Then, it queries $\mathcal{C}^{\mathsf{prf}}_f$ for inputs $(\mathsf{Out}_1 \| m^{(2)} \| m^{(3)}, a, b)$ and receives the value Out_2.
- It returns the value Out_2.

We note that the two first bullets permits to generate $G(\mathsf{sk}, m, a, b)$ computing $f(\mathsf{sk}, f(\mathsf{sk}, m^{(1)} \| m^{(3)}, 0, 256) \| m^{(2)} \| m^{(3)}, a, b)$. This step is repeated up to a total of q queries from $\mathcal{A}_g$, with a and b fixed.

At some point, $\mathcal{A}_g$ halts and outputs a guess d of the bit b. The prf-adversary $\mathcal{A}_f$ chooses its guess b' as $b' = d$ and forwards it to $\mathcal{C}^{\mathsf{prf}}_f$, which verifies if $b = b'$.

We analyze this simulation. Recall that the challenger responded either with a random value (if its internal bit b was set to 0) or with the output of the function $f(\mathsf{sk}, m^{(1)} \| m^{(3)}, 0, 256)$ (if its internal bit was set as 1).

Thus, the output Out_2 matches either the output of a random function or the output of the function $G(\mathsf{sk}, m, a, b)$. So the prf-adversary $\mathcal{A}_f$ simulates perfectly a prf-challenger of G. Thus, we have:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{prf}}_{f}(\mathcal{A}_f) &= \big|\Pr[\mathcal{A}_f \to 1 \mid b = 1] - \Pr[\mathcal{A}_f \to 1 \mid b = 0]\big| \\
&= \big|\Pr[b = b' | b = 1] - \Pr[b = b' | b = 0]\big| \\
&= \big|\Pr[b = d | b = 1] - \Pr[b = d | b = 0]\big| \\
&= \big|\Pr[d' = d | b = 1] - \Pr[d' = d | b = 0]\big| \\
&= \big|\Pr[\mathcal{A}_G \to 1 \mid b = 1] - \Pr[\mathcal{A}_G \to 1 \mid b = 0]\big| \\
&= \big|\mathsf{Adv}^{\mathsf{prf}}_{G}(\mathcal{A}')\big|.
\end{aligned}$$

We use the generic result specified in A.1 to reduce the mac-secure to the prf-secure of the function G.

Theorem 7. [$\mathsf{Mac}-$**security for** $\mathbf{G}_{\mathsf{tuak}}$.] *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* Mac*-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* $\mathsf{Adv}^{\mathsf{mac}}_{G_{\mathsf{tuak}}}(\mathcal{A})$. *Then there exists a* $(t' \approx 2 \cdot (t + O(n + d)), q' = 2 \cdot q)$*-adversary* $\mathcal{A}$*' with an advantage* $\mathsf{Adv}^{\mathsf{prf}}_{f}(\mathcal{A}')$ *of winning against the pseudorandomness of* f *such that:*

$$\mathsf{Adv}^{\mathsf{mac}}_{G_{\mathsf{tuak}}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{prf}}_{f}(\mathcal{A}') + \frac{1}{2^n}.$$

Indistinguishability of TUAK **algorithms.** We begin by reducing the $\mathsf{ind}-\mathsf{cpa}$-security of G_{tuak} to the prf-security of the function f. This implies the $\mathsf{ind}-\mathsf{cpa}$-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 8. [$\mathsf{ind-cpa-securityofG_{tuak}}$.] *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* $f : \{0,1\}^\kappa \times \{0,1\}^d \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^m$ *be the two functions specified in Section C. Consider a* (t,q)*-adversary* $\mathcal{A}$ *against the* $\mathsf{ind-cpa}$*-security of the function* G*, running in time* t *and making at most* q *queries to its challenger. Denote the advantage of this adversary as* $\mathsf{Adv}^{\mathsf{ind-cpa}}_{G_{\mathsf{tuak}}}(\mathcal{A})$*. Then there exists 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:*

$$\mathsf{Adv}^{\mathsf{ind-cpa}}_{G_{\mathsf{tuak}}}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{prf}}_f(\mathcal{A}'),$$

Proof. To prove the $\mathsf{ind-cpa}$-security of G, we reduce this security to the prf-security of G which is defined in the proof of the theorem 7. We show that a prf-adversary $\mathcal{A}$ of G can simulate the $\mathsf{ind-cpa}$-challenger $\mathcal{C}^{\mathsf{ind-cpa}}_G$. A such prf-adversary behaves as follows. At first, the challenger $\mathcal{C}^{\mathsf{prf}}_G$ chooses a private key K and one random bit $b \in \{0,1\}$. If $b = 0$, he assigns f to a random function and if $b = 1$, he assigns f to the specific function G. For each query $(M^{\{i\}}, a, b)$ with a fixed (a, b) from the $\mathsf{ind-cpa}$-adversary $\mathcal{A}$' to the prf-adversary $\mathcal{A}$ the latter forwards each one to $\mathcal{C}^{\mathsf{prf}}_G$. The answer $f(K, M^{\{i\}}, a, b)$ is sent to $\mathcal{A}$ which forwards it to $\mathcal{A}$'. Then, $\mathcal{A}$' sends a specific query containing two values (M_0, M_1) to $\mathcal{A}$The latter chooses randomly a bit d and forwards (M_d, a, b) to the prf-challenger. As usual, this challenger sends $f(K, M_d, a, b)$ to $\mathcal{A}$ which forwards it to $\mathcal{A}$'.

The goal of the $\mathsf{ind-cpa}$-adversary is to find the bit d chosen by the $\mathcal{A}$. To do so, it can ask again some queries $(M^{\{i\}}, a, b)$ as previously. Finally, it sends its guessing d' to $\mathcal{A}$. Upon receiving this guessing, it chooses its guessing b' of b as follows: if $d = d'$, it chooses $b' = 1$, else $b' = 0$.

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{prf}}_G(\mathcal{A}) &= \big|\Pr[b = b' | f \xleftarrow{\$} G(K, ., ., .), K \xleftarrow{\$} \{0,1\}^\kappa] \\
&\qquad - \Pr[b = b' | f \xleftarrow{\$} \mathsf{R}]\big| \\
&= \big|\Pr[b = b' | b = 1] - \Pr[b = b' | b = 0]\big| \\
&= \big|\Pr[b' = 1 | b = 1] - \Pr[b' = 0 | b = 0]\big| \\
&= \big|\Pr[d' = d | b = 1] - \Pr[d' \neq d | b = 0]\big| \\
&= \big|\mathsf{Adv}^{\mathsf{ind-cpa}}_G(\mathcal{A}') + \tfrac{1}{2} - \tfrac{1}{2}\big| \\
&= \big|\mathsf{Adv}^{\mathsf{ind-cpa}}_G(\mathcal{A}')\big|
\end{aligned}$$

In the last equality, $\Pr[d' = d | b = 1]$ is the probability that the $\mathsf{ind-cpa}$-adversary correctly guess the bit d which is $\mathsf{Adv}^{\mathsf{ind-cpa}}_G(\mathcal{A}') + 1/2$ and $\Pr[d' \neq d | b = 0]$ which is equal to $1/2$ since when $b = 0$, G is a random function, that is its output is chosen independently from its inputs. Consequently, it is not related to its inputs and the adversary cannot guess correctly the bit d.

Updated MILENAGE algorithms security In order to prove the unforgeability and indistinguishability properties 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:

$$f(K, x, y) = \mathsf{AES}_K(x \oplus y),$$

with $x, y \in \{0,1\}^{128}$, $K \in \{0,1\}^\kappa$. Contrary to the TUAK algorithms, the MILENAGE algorithms do not behave as the same way but as two different ways. Let the construction G_{mil1}, the generalization of the functions $\mathcal{F}_1$, $\mathcal{F}_1^*$, $\mathcal{F}_2$, $\mathcal{F}_3$, $\mathcal{F}_4$ and G_{mil2} the generalization of the functions $\mathcal{F}_5$ and $\mathcal{F}_5^*$ as follows:

$$
\begin{aligned}
G_{\mathsf{mil1}}(\mathsf{sk_C}, \mathsf{inp}_1, 0, 127) &= \mathcal{F}_1(\mathsf{sk_{op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Sqn}, \mathsf{Res_S}, \mathsf{AMF}),\\
G_{\mathsf{mil1}}(\mathsf{sk_C}, \mathsf{inp}_2, 0, 127) &= \mathcal{F}_1^*(\mathsf{sk_{op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Sqn}, \mathsf{Res_S}, \mathsf{AMF}),\\
G_{\mathsf{mil1}}(\mathsf{sk_C}, \mathsf{inp}_3, 0, 47) &= \mathcal{F}_2(\mathsf{sk_{op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Sqn}, \mathsf{Res_S}, \mathsf{AMF}),\\
G_{\mathsf{mil1}}(\mathsf{sk_C}, \mathsf{inp}_4, 0, 127) &= \mathcal{F}_3(\mathsf{sk_{op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Sqn}, \mathsf{Res_S}, \mathsf{AMF}),\\
G_{\mathsf{mil1}}(\mathsf{sk_C}, \mathsf{inp}_5, 0, 127) &= \mathcal{F}_4(\mathsf{sk_{op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Sqn}, \mathsf{Res_S}, \mathsf{AMF}),\\
G_{\mathsf{mil2}}(\mathsf{sk_C}, \mathsf{inp}_6, 0, 47) &= \mathcal{F}_5(\mathsf{sk_{op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Res_S}),\\
G_{\mathsf{mil2}}(\mathsf{sk_C}, \mathsf{inp}_6, 64, 111) &= \mathcal{F}_5^*(\mathsf{sk_{op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Res_S}),
\end{aligned}
$$

with:

$\forall i \in \{1, ..., 5\}, \mathsf{inp}_i = \mathsf{sk_{op}} \| \mathsf{R} \| (\mathsf{Sqn} \| \mathsf{AMF}) \| \mathsf{Res_S} \| \mathsf{c}_i \| \mathsf{r}_i$.

$\mathsf{inp}_6 = \mathsf{sk_{op}} \| \mathsf{R} \| \mathsf{Res_S} \| \mathsf{c}_6 \| \mathsf{r}_6$,

These both constructions are constructed as follows:

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

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

We express the security properties of the generalizations G_{mil1} and G_{mil2} under the prf-security of the function 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 c_1 and c_2 are two constants depending only on this model. In other words, we assume that the best attacks are either a exhaustive key search or a linear cryptanalysis.

Pseudorandomness and Unforgeability 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 Mac-security of each MILENAGE algorithm.

Theorem 9. $[\mathsf{prf} - \mathbf{securityfor}\mathbf{G}_{\mathsf{mil1}}\mathbf{and}\mathbf{G}_{\mathsf{mil2}}.]$ *Let* $G_{\mathsf{mil1}}, G_{\mathsf{mil2}} : \{0,1\}^\kappa \times \{0,1\}^d \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^n$ *and* $f : \{0,1\}^\kappa \times \{0,1\}^{128} \times \{0,1\}^{128} \to \{0,1\}^{128}$ *be the two functions specified above. Consider a* (t,q)*-adversary* $\mathcal{A}$ *against the* prf*-security of the function* G_{mil1} *(respectively* G_{mil2}*), running in time* t *and making at most* q *queries to its challenger. Denote the advantage of this adversary as* $\mathsf{Adv}^{\mathsf{prf}}_{G_{\mathsf{mil1}}}(\mathcal{A})$ *(respectively* $\mathsf{Adv}^{\mathsf{prf}}_{G_{\mathsf{mil2}}}(\mathcal{A})$*). 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:*

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

Proof. We construct the adversary $\mathcal{A}_f$ using a prf-adversary $\mathcal{A}_{G_1}$ (respectively $\mathcal{A}_{G_2}$). The latter uses $\mathcal{A}_f$ as a challenger for a prf-game $\mathbb{G}_{\mathsf{prf}(f)}$ and can only communicate with $\mathcal{A}_f$ whereas $\mathcal{A}_f$ has access to a challenger for f. To begin with, the challenger $\mathcal{C}_f^{\mathsf{prf}}$ chooses a bit b and a private $\mathsf{sk} \in \{0,1\}^\kappa$. If $b = 0$, it assigns f to a random function and if $b = 1$, it assigns f to the specific internal function.

The adversary $\mathcal{A}_f$ waits for queries from $\mathcal{A}_G$ of the form (m, a, b), with $m = m^{(1)} \| m^{(2)} \| m^{(3)} \| m^{(4)} \| m^{(5)} \in \{0,1\}^d$, and $a, b \in \{0,1\}^t$ and responds as follows:

– It queries its challenger $\mathcal{C}_f^{\mathsf{prf}}$ for inputs $(m^{(1)}, 0^{128})$ and receives the value Out_1.
– Then, it computes $\mathsf{Top}_C = m^{(1)} \oplus Out_1$ and it queries $\mathcal{C}_f^{\mathsf{prf}}$ for inputs $(\mathsf{Out}_1, m^{(2)})$ and receives the value Out_2.
– It queries $(m^{(4)}, \mathsf{Out}_2 \oplus rot(\mathsf{Out}_1 \oplus (m^{(3)} \| m^{(3)}), m^{(5)}))$ (respectively $(m^{(4)}, rot(\mathsf{Out}_1 \oplus \mathsf{Out}_2, m^{(5)}))$) and receives the value Out_3.
– It returns the value $\lfloor \mathsf{Out}_1 \oplus \mathsf{Out}_3 \rfloor_{a,b}$.

This step is repeated up to a total of q queries from $\mathcal{A}_{G_1}$ (respectively $\mathcal{A}_{G_2}$), with a and b fixed.

At some point, $\mathcal{A}_{G_1}$ (respectively $\mathcal{A}_{G_2}$) halts and outputs a guess d of the bit b. The prf-adversary $\mathcal{A}_f$ chooses its guess b' as $b' = d$ and forwards it to $\mathcal{C}_f^{\mathsf{prf}}$, which verifies if $b = b'$.

We analyze this simulation. Recall that the challenger responded either with a random value (if its internal bit b was set to 0) or with the output of the function $f(\mathsf{sk}, ., ., .)$ (if its internal bit was set as 1).

Thus, the output Out_3 matches either the output of the output of the function $G_1(\mathsf{sk}, m, a, b)$ (respectively $G_2(\mathsf{sk}, m, a, b)$) or a random function (indeed, the combination of two random functions by a boolean addition gives a random function). So the prf-adversary $\mathcal{A}_f$ simulates perfectly a prf-challenger of G_1 (respectively G_2). Thus, we have:

$$\begin{aligned}
\mathsf{Adv}_f^{\mathsf{prf}}(\mathcal{A}_f) &= \Big| \Pr[\mathcal{A}_f \to 1 \mid b = 1] - \Pr[\mathcal{A}_f \to 1 \mid b = 0] \Big| \\
&= \Big| \Pr[b = b' | b = 1] - \Pr[b = b' | b = 0] \Big| \\
&= \Big| \Pr[b = d | b = 1] - \Pr[b = d | b = 0] \Big| \\
&= \Big| \Pr[d' = d | b = 1] - \Pr[d' = d | b = 0] \Big| \\
&= \Big| \Pr[\mathcal{A}_{G_1} \to 1 \mid b = 1] - \Pr[\mathcal{A}_{G_1} \to 1 \mid b = 0] \Big| \\
&= \mathsf{Adv}_{G_1}^{\mathsf{prf}}(\mathcal{A}').
\end{aligned}$$

(similar computation for G_2).

We use the generic result specified in A.1 to reduce the mac-secure to the prf-secure of the function G_1 (resp. G_2).

Theorem 10. [$\mathsf{Mac}-$**security for** $\mathbf{G}_{\mathsf{mil1}}$ **and** $\mathbf{G}_{\mathsf{mil2}}$.]
Let $G_{\mathsf{mil1}}, G_{\mathsf{mil2}} : \{0,1\}^\kappa \times \{0,1\}^d \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^n$ *and* $f : \{0,1\}^\kappa \times \{0,1\}^{128} \times \{0,1\}^{128} \to \{0,1\}^{128}$ *be the two functions specified above. Consider a* (t,q)*-adversary* $\mathcal{A}$ *against the* Mac*-security of the function* G_{mil1} *(respectively* G_{mil2}*), running in time* t *and making at most* q *queries to its challenger. Denote the advantage of this adversary as* $\mathsf{Adv}_{G_{\mathsf{mil1}}}^{\mathsf{mac}}(\mathcal{A})$ *(respectively* $\mathsf{Adv}_{G_{\mathsf{mil2}}}^{\mathsf{mac}}(\mathcal{A})$*). Then there exists a* $(t' \approx 3{\cdot}(t{+}O(n{+}d)), q' = 3{\cdot}q)$*-adversary* $\mathcal{A}'$ *with an advantage* $\mathsf{Adv}_f^{\mathsf{prf}}(\mathcal{A}')$ *of winning against the pseudorandomness of* f *such that:* $\mathsf{Adv}_{G_{\mathsf{mil1}}}^{\mathsf{mac}}(\mathcal{A}) = \mathsf{Adv}_{G_{\mathsf{mil2}}}^{\mathsf{mac}}(\mathcal{A}) \leq \mathsf{Adv}_f^{\mathsf{prf}}(\mathcal{A}') + \frac{1}{2^n}$.

Indistinguishability of MILENAGE **algorithms.** We begin by reducing the $\mathsf{ind}-\mathsf{cpa}$-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 11. [$\mathsf{ind}-\mathsf{cpa}-$**security of** $\mathbf{G}_{\mathsf{mil1}}$ **and** $\mathbf{G}_{\mathsf{mil2}}$.]
Let $G_{\mathsf{mil1}}, G_{\mathsf{mil2}} : \{0,1\}^\kappa \times \{0,1\}^d \times \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^n$ *and* $f : \{0,1\}^\kappa \times \{0,1\}^{128} \times \{0,1\}^{128} \to \{0,1\}^{128}$ *be the two functions specified in Section C. Consider a* (t,q)*-adversary* $\mathcal{A}$ *against the* $\mathsf{ind}-\mathsf{cpa}$*-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* $\mathsf{Adv}_{G_{\mathsf{mil1}}}^{\mathsf{ind-cpa}}(\mathcal{A})$ *(resp.* $\mathsf{Adv}_{G_{\mathsf{mil2}}}^{\mathsf{ind-cpa}}(\mathcal{A})$*). Then there exists a* $(t' \approx 3 \cdot t, q' = 3 \cdot q)$*-adversary* $\mathcal{A}$*' with an advantage* $\mathsf{Adv}_f^{\mathsf{prf}}(\mathcal{A}')$ *of winning against the pseudorandomness of* f *such that:*

$$\mathsf{Adv}_{G_{\mathsf{mil1}}}^{\mathsf{ind-cpa}}(\mathcal{A}) = \mathsf{Adv}_{G_{\mathsf{mil2}}}^{\mathsf{ind-cpa}}(\mathcal{A}) = \mathsf{Adv}_f^{\mathsf{prf}}(\mathcal{A}'),$$

Proof. To prove the $\mathsf{ind}-\mathsf{cpa}$-security of G_1 (resp. G_2, we reduce this security to the prf-security of G_1 (resp. G_2) and then to the prf-security of the function f using the theorem 9 . The first reduction follows the exact lines of the proof of the prf-security of the TUAK algorithms and we obtain:

$$\mathsf{Adv}^{\mathsf{ind}-\mathsf{cpa}}_{G_1}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{ind}-\mathsf{cpa}}_{G_2}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{prf}}_{f}(\mathcal{A}'),$$

4.3 Narrow-Forward Privacy is Impossible

Our variant of AKA preserves the structure of the original protocol, but also provably attains wide-weak client unlinkability. In this section we show that this degree of client-unlinkability is optimal with respect to the structure of AKA. In particular, narrow-forward privacy is impossible.

Our result covers similar ground as that by Paise and Vaudenay [21], as we address protocols with mutual authentication. We extend the impossibility result to symmetric-key AKE protocols which also use public-key primitives. We also explain why the original impossibility result in [21] is imprecise, and presents some problems.

The result of [21]. Paise and Vaudenay showed an impossibility result for *authentication* protocols – rather than AKE; the extension from one environment to the other is, however, easy. In the terminology of our paper, [21] proved that server-authentication essentially precludes narrow-forward client-unlinkability. Their attack follows these steps: (1) the adversary $\mathcal{A}$ creates two clients; (2) $\mathcal{A}$ runs an honest protocol session between one of them (chosen uniformly at random depending on a secret bit b) and the server, but stops the last message from the server to the client; (3) $\mathcal{A}$ corrupts both clients, learning their long-term state; (4) $\mathcal{A}$ distinguishes between the clients by simulating the protocol with the intercepted message.

However, this attack makes a tacit assumption on the client's behaviour, namely that if a session is aborted, the state is not updated or that it is updated in a consistent way, depending on the client's internal state. Say that upon an abort, the client reverts to a random state; assumming that the adversary cannot access the very short time-frame in which the reversion to random is done, $\mathcal{A}$ only gets the random state in response. Simulating the protocol with the received message will not match that state, thus reducing the adversary's success probability to $\frac{1}{2}$.

Another way to bypass this result is to update the client state before the "last message" is sent to the client; if such an update is done at every execution, the attack presented in [21] fails. This is, however, a rather artificial twist: indeed, mutual authentication implies that the prover *must* somehow identify the server's state as "valid" *before* it reaches a state which precludes it from verifying the server's authentication.

Two new attacks. In the AKA protocol, it is the server which first authenticates to the client. The values used in authentication are the sequence number $\mathsf{Sqn}_{\mathsf{Op},\mathsf{C}}$ and the nonce R. The value $\mathsf{Sqn}_{\mathsf{Op},\mathsf{C}}$ is ephemeral, being updated at every session; however, it is a long-term state, and compatible with the client's own state $\mathsf{Sqn}_{\mathsf{C}}$. In particular, corrupting a client yields $\mathsf{Sqn}_{\mathsf{C}}$ allowing the verifier to link the client with the corresponding $\mathsf{Sqn}_{\mathsf{Op},\mathsf{C}}$ value.

For a better comprehension of our attacks and their impact, we define the following notations: we divide a party's state (for both clients and operators) into a *static* state $\mathsf{stat.st}_{\mathsf{P}}$ and an ephemeral state $\mathsf{eph.st}_{\mathsf{P}}$. Thus, an operator's static state may contain operator-specific information, such as the secret key for a PKE scheme, but it will also include state shared with clients, i.e. $\mathsf{stat.st}_{\mathsf{Op},\mathsf{C}}$ for a client C. The same for the ephemeral state $\mathsf{eph.st}_{\mathsf{Op},\mathsf{C}}$ which for the AKA protocol consists of the sequence number $\mathsf{Sqn}_{\mathsf{Op},\mathsf{C}}$.

We propose the following attack:

- The adversary $\mathcal{A}$ creates two clients C and C' with the same operator Op and the same location LAI.
- $\mathcal{A}$ uses DrawCl on C, C' and receives the handle VC, corresponding to either C (if the hidden bit $b = 0$), or C' (otherwise).
- $\mathcal{A}$ runs an honest execution between the server S at LAI and the client VC until $\mathcal{A}$ receives the message $\mathsf{R}, \mathsf{Autn} = (\mathsf{Sqn}_{\mathsf{Op},\mathsf{C}} \oplus \mathsf{AK}) \| \mathsf{AMF} \| \mathsf{Mac}_{\mathsf{S}}$ from the server. Denote $\mathsf{Autn}[1] := \mathsf{Sqn}_{\mathsf{Op},\mathsf{C}} \oplus \mathsf{AK}$, $\mathsf{Autn}[2] = \mathsf{AMF}$, and $\mathsf{Autn}[3] := \mathsf{Mac}_{\mathsf{S}}$.

– $\mathcal{A}$ corrupts C and learns $\mathsf{Sqn_C}$, $\mathsf{sk_C}$, and $\mathsf{sk_{op}}$.
– By using the values $\mathsf{R}, \mathsf{sk_C}$, and $\mathsf{sk_{op}}$, the adversary computes a value $\mathsf{AK_C}$ and retrieves $\mathsf{Sqn}^* := \mathsf{Autn}[1] \oplus \mathsf{AK_C}$. Note that if $b = 0$, then $\mathsf{AK_C} = \mathsf{AK}$ and $\mathsf{Sqn}^* = \mathsf{Sqn_{Op,C}}$, while if $b = 1$, then $\mathsf{Sqn}^* \neq \mathsf{Sqn_{Op,C'}}$ with overwhelming probability.
– The adversary verifies $\mathsf{Mac_S}$, on input $(\mathsf{sk_C}, \mathsf{sk_{op}}, \mathsf{Sqn}^*)$. If this verification succeeds, the adversary outputs a guess $d = 0$ for the bit b; else, it outputs 1.

For the analysis note that with overwhelming probability $\mathsf{Mac_S}$ will not verify if it was computed for $\mathsf{C'}$, i.e. $f_1(\mathsf{sk_{C'}}, \mathsf{sk_{op}}, \mathsf{R}, \mathsf{AMF}, \mathsf{Sqn_{Op,C'}}) \neq f_1(\mathsf{sk_C}, \mathsf{sk_{op}}, \mathsf{R}, \mathsf{AMF}, \mathsf{Sqn}^*)$. The key vulnerability here is that, while $\mathsf{Sqn_{Op,C}}$ is never sent in clear, the masking authentication key AK only depends on the client's static state. We also use the fact that the validity of the sequence number is confirmed by the value $\mathsf{Mac_S}$.

While the latter factor, namely the validity of $\mathsf{Mac_S}$ certainly helps an attacker, our second attack (a variation of the first one) does not use the MAC value at all. The attack is run exactly in the same way, until we reach the final step. At that point:

– $\mathcal{A}$ compares the obtained value Sqn^* with the recovered sequence number of C, namely $\mathsf{Sqn_C}$, verifying if $|\mathsf{Sqn}^* - \mathsf{Sqn_{Op,C}}| \leq \Delta$. Note that in the actual attack presented above, the client's state $\mathsf{Sqn_C}$ should be exactly equal to the operator's state with respect to that client; however, our attack is even stronger in the sense that we do not need to control the executions of the protocol in order to obtain exact equality.

Analysis and Impact. Since the original AKA protocol is not even weak-client-unlinkable, it is not surprising that this protocol is not narrow-forward unlinkable either. However, the same attack works on our variant of the protocol and indeed, on any other extension or improvement of the original procedure which retains the characteristic of exchanging a message of the type $f(\mathsf{eph.st_{Op,C}}, \mathsf{stat.st_{Op,C}}, X)$ in the presence of a function $Match$; or exchanging that same message together with a message $g(\mathsf{eph.st_{Op,C}}, \mathsf{stat.st_{Op,C}}, Y)$, such that:

– f is reversible and takes as input $\mathsf{eph.st_{Op,C}}, \mathsf{stat.st_{Op,C}} = \mathsf{stat.st_C}$, and a set X of publicly-known variables, giving arbitrary values in the set $\{0,1\}^*$[7];
– $Match$ takes as input two ephemeral state values $\mathsf{eph.st_{C'}}$ and $\mathsf{eph.st_{Op,C}}$ and it outputs a boolean value: 1 if $\mathsf{C} = \mathsf{C'}$ and 0 otherwise[8];
– g takes as input the state values $\mathsf{eph.st_{Op,C}}, \mathsf{stat.st_C}$ and a set Y of public values, and which has the property that, for randomly chosen x and $\mathsf{stat.st_{C'}}$ it holds that $g(x, \mathsf{stat.st_{Op,C'}}, Y) \neq g(\mathsf{eph.st_{Op,C}}, \mathsf{stat.st_{Op,C}}, Y)$[9].

5 Practical considerations

In this section, we discuss some of our design choices for the improvement we propose of the AKA protocol. We also provide a detailed analysis of our countermeasures and their intuitive effects in Appendix B.

As opposed to the proposal of van den Broek et al. [7], we opted to continue using (TMSI, LAI) tuples for the identification phase of the protocol. This infrastructure is maintained strictly by servers, with no operator contribution; thus it is efficient and inexpensive. Moreover, TMSI values and their correspondence to the client's IMSI is easy to find. In our proposal, we bypass IMSI catchers attacks by never sending IMSIs in clear, and we add a symmetric authentication step in the encryption, thus precluding the client-unlinkability attack we found against the AKA variant of Arapinis et al. [20]. For the encryption, we use an IND-CCA public-key encryption scheme, we require a minimum PKI, only for operators. A client only stores the public key (and certificate) of the operator it subscribes to, thus minimizing key-management problems. In the TMSI

[7] In our previous example, this is the string Autn1, which depends on $\mathsf{eph.st_{Op,C}} = \mathsf{Sqn_{Op,C}}$, on $\mathsf{stat.st_{Op,C}} = (\mathsf{sk_C}, \mathsf{sk_{op}})$, and on the random value R which is public.
[8] In our case, the $Match$ function returns 1 if and only if $|\mathsf{Sqn_{Op,C}} - \mathsf{Sqn_{C'}}| \leq \Delta$.
[9] In our example, this function is f_1, and the output value is $\mathsf{Mac_S}$.

reallocation step, we add an implicit authentication step, preventing Denial-of-Service-based linkability. We also add a freshness index, which prevents replays of challenges based on old sequence numbers.

We do specify, however, that our variant can only guarantee client-unlinkability if the *size* of the TMSI is equal to the length of the output of the PKE scheme. This is a non-trivial requirement, since servers are expected to keep track of all the TMSIs they issue; while using a shorter TMSI does not leak anything about the IMSI value, it does allow mass-surveillance organisms to track users down by distinguishing between the length of the encrypted IMSI as opposed to the TMSI length. On the positive side, servers may store TMSI values for a shorter while, since as soon as the user leaves the area, the TMSI is no longer useful.

Moreover, we recommend using a field of 32 bits for the index values $\mathsf{idx}_{\mathsf{C}}$, $\mathsf{idx}_{\mathsf{Op,C}}$. In fact, every time a session is aborted, the index(s) is (are) increased. The only way to replay a challenge is to previously drop 2^{32} successive authentication challenges, which is in our opinion hard to do. We require that the size of all the variables (except the network variables $\mathsf{Op}_{\mathsf{Id}}$ and $\mathsf{Res}_{\mathsf{S}}$) is: 96 bits for a 64-bit security bound, 128 bits for 96-bit security bound and 154 bits for 128-bit security bound.

We make the assumption that clients are aware of their current LAI, and thus avoid client-tracking by means of an itinerary. This is not a very strong assumption, since mobile devices are often equipped to detect their LAI. Finally, we bypass distinguishing attacks that exploit the re-synchronization phase by ensuring that sequence numbers cannot be desynchronized (and replays of challenges using old sequence numbers are prevented). Keeping in mind the practical requirement of minimizing the communication between servers and operators, our variant ensures that operators are contacted only in case the protocol is abnormally run or an adversary is detected. We also simplify the rather complex AKA structure, including only three communication phases rather than five. We depict our countermeasures and discuss them more in detail in Appendix B.

Finally, we require to restrict the batch of authentication vectors at only one vector if the last message (sent from the server to update the operator sequence number) can be dropped.

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 (TSG) SA; 3G Security; Security Architecture. TS 33.102, 3rd Generation Partnership Project (3GPP), June 2013.
4. 3GPP. 3rd Generation Partnership Project; Technical Specification Group Services ans System Aspects; Security related network functions (Release 12). TS 43.020, 3rd Generation Partnership Project (3GPP), June 2014.
5. David A. McGrew and John Viega. The Security and Performance of the Galois/Counter Mode of Operation (Full Version). *IACR Cryptology ePrint Archive*, 2004:193, 2004.
6. D.Strobel. IMSI Catcher. In *2007, Seminar Work, Ruhr-Universitat Bochum*, 2007.
7. Fabian van den Broek and Roel Verdult and Joeri de Ruiter. Defeating IMSI Catchers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, USA, October 12-6, 2015*, pages 340–351, 2015.
8. 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.
9. Jens Hermans and Andreas Pashalidis and Frederik Vercauteren and Bart Preneel. A New RFID Privacy Model. In V. Atluri and C. Diaz, editors, *Esorics*, volume 6879, pages 568–587, 2011.
10. Jens Hermans and Andreas Pashalidis and Frederik Vercauteren and Bart Preneel. A New RFID Privacy Model. In *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*, pages 568–587, 2011.
11. Michael Burrows and Martín Abadi and Roger M. Needham. A Logic of Authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990.

12. 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.
13. Mihir Bellare and Joe Kilian and Phillip Rogaway. The Security of the Cipher Block Chaining Message Authentication Code. *J. Comput. Syst. Sci.*, 61(3):362–399, 2000.
14. Mihir Bellare and Phillip Rogaway. Entity Authentication and Key Distribution. In D. R. Stinson, editor, *Advances in Cryptology - CRYPTO '93*, volume 773 of *LNCS*, pages 232–249. Springer, 1993.
15. Mihir Bellare and Ran Canetti and Hugo Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 419–428, 1998.
16. 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.
17. Muxiang Zhang. Provably-Secure Enhancement on 3GPP Authentication and Key Agreement Protocol. *IACR Cryptology ePrint Archive*, 2003:92, 2003.
18. 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.
19. Myrto Arapinis and Loretta Ilaria Mancini and Eike Ritter and Mark Ryan. Privacy through Pseudonymity in Mobile Telephony Systems. In *21st Annual Network and Distributed System Security Symposium, NDSS*, 2014.
20. Myrto Arapinis and Loretta Ilaria Mancini and Eike Ritter and Mark Ryan and Nico Golde and Kevin Redon and Ravishankar Borgaonkar. New privacy issues in mobile telephony: fix and verification. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 205–216, 2012.
21. Radu-Ioan Paise and Serge Vaudenay. Mutual Authentication in RFID: Security and Privacy. In *Proc. on the 3^{rd} ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 292–299. ACM, 2008.
22. Ran Canetti and Hugo Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 337–351, 2002.
23. Serge Vaudenay. On Privacy Models for RFID. In *ASIACRYPT '07*, volume 4883, pages 68–87, 2007.
24. Ulrike Meyer and Susanne Wetzel. A man-in-the-middle attack on UMTS. In *Proceedings of the 2004 ACM Workshop on Wireless Security, Philadelphia, PA, USA, October 1, 2004*, pages 90–97, 2004.
25. Victor Shoup. A Proposal for an ISO Standard for Public Key Encryption. Cryptology ePrint Archive, Report 2001/112, 2001. `http://eprint.iacr.org/`.
26. Zahra Ahmadian and Somayeh Salimi and Ahmad Salahi. New attacks on UMTS network access. In *2009 Wireless Telecommunications Symposium, WTS 2009, Prague, Czech Republic, April 22-24, 2009*, pages 1–6, 2009.

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 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 $\mathsf{Rand} : \{0,1\}^d \rightarrow \{0,1\}^n$. 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 $\mathsf{Rand} : \{0,1\}^d \rightarrow \{0,1\}^n$:

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

Definition 11. *([**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.*

IND-CPA security. A scheme is considered secure in terms of indistinguishability (against chosen message attacks) if no adversary can learn any information on the input of the scheme given the output. This is formalized in terms of the following security game (denoted $\mathbb{G}^{\mathsf{ind-cpa}}$):

1. The challenger $\mathcal{C}_f^{\mathsf{ind-cpa}}$ chooses a key $K \in \{0,1\}^\kappa$.
2. The adversary $\mathcal{A}$ chooses and sends q_1 messages x to the challenger, which returns $f(K,x)$.
3. $\mathcal{A}$ chooses two messages x_0 and x_1 and sends them to $\mathcal{C}_f^{\mathsf{ind-cpa}}$.
4. The challenger chooses a bit b and returns (K, x_b) to the adversary.
5. Upon receiving $f(K,x_b)$, the adversary can adaptively query the challenger with at most q_2 messages $x \notin \{x_0, x_1\}$, to which the challenger responds with $f(K,x)$.
6. Finally, the adversary halts and outputs a guess d of the bit b used by the challenger.

Definition 12. *([**General security of ind-cpa-scheme.**]) A scheme f is considered as $\mathsf{ind-cpa}$-cpa-secure by indistinguishability if any adversary $\mathcal{A}$ running in time t and making at most q queries to its challenger $\mathcal{C}_f^{\mathsf{ind-cpa}}$, given an encryption of a message randomly chosen from a two-element message space, cannot distinguish efficiently the encryption of one of both messages. We can evaluate the $\mathsf{ind-cpa}$-advantage of a such adversary, denoted $\mathsf{Adv}_f^{\mathsf{ind-cpa}}(\mathcal{A})$:*

$$\mathsf{Adv}_f^{\mathsf{ind-cpa}}(\mathcal{A}) = Pr[\mathcal{A} \to 1 | b \leftarrow \{0,1\}, K \leftarrow \{0,1\}^\kappa, (x_0,x_1) \in \{0,1\}^d \times \{0,1\}^d, \mathcal{C} \to f(K,x_b)].$$

MAC security. A scheme is considered secure in terms of unforgeability (against chosen message attacks) if no adversary can forge an acceptable message authentication code for any chosen message. This is formalized in terms of the following security game (denoted $\mathbb{G}^{\mathsf{Mac}}$):

1. The challenger $\mathcal{C}_f^{\mathsf{Mac}}$ chooses a key $K \in \{0,1\}^\kappa$.
2. The adversary $\mathcal{A}$ chooses and sends q messages x to the challenger, which returns $f(K,x)$.
3. The adversary tries to forge an acceptable output σ for a chosen input x which was no a previous query and sends the couple (x,σ) to the challenger.
4. The challenger verifies if the message x was not previously requested and if $f(K,x) = \sigma$. The adversary wins this game if the both conditions are accepted. Otherwise, the adversary loses the game.

We define the adversary's advantage with respect to this game as:

$$\mathsf{Adv}_f^{\mathsf{mac}}(\mathcal{A}) = Pr[\mathcal{A}wins].$$

Definition 13. *([General security of mac-scheme.]) A MAC construction (f, ϑ) is considered as* mac*-secure by unforgeability if any adversary $\mathcal{A}$ running in time t and making at most q queries to its challenger $\mathcal{C}_f^{\mathsf{mac}}$, cannot forge efficiently any couple (message, mac) no-forged by $\mathcal{C}_f^{\mathsf{mac}}$. We can evaluate the mac-advantage We can evaluate the* $\mathsf{ind} - \mathsf{cpa}$*-advantage of a such adversary, denoted $\mathsf{Adv}_f^{\mathsf{mac}}(\mathcal{A})$:*

$$\mathsf{Adv}_f^{\mathsf{mac}}(\mathcal{A}) = \Pr[\vartheta(x_{q+1}, \tau_{q+1}) = 1 | K \xleftarrow{\$} \{0,1\}^\kappa, \\ \forall i \in \{1, .., q\}, \mathcal{C} \to (x_i, \tau_i), \mathcal{A} \to (x_{q+1}, \tau_{q+1})].$$

Generic Results. As mentioned in [13], pseudorandom functions make good message authentication codes. The authors have determined the exact security of a such reduction by the following proposal:

Proposition 1. *Let $f : \{0,1\}^\kappa * \{0,1\}^d \to \{0,1\}^s$ be a family of functions. Consider a (t, q)-adversary $\mathcal{A}$ against the prf-security of the function f, running in time t and making at most q queries to its challenger. Denote the advantage of a such adversary $\mathsf{Adv}_f^{\mathsf{prf}}(\mathcal{A})$. then, there are a $(t' \sim t + O(s + d), q' = q)$-adversary $\mathcal{A}$' against the mac-security of the function f with an advantage $\mathsf{Adv}_f^{\mathsf{mac}}(\mathcal{A}')$ such as:*

$$\mathsf{Adv}_f^{\mathsf{mac}}(\mathcal{A}) \leq \mathsf{Adv}_f^{\mathsf{prf}}(\mathcal{A}') + \frac{1}{2^s}.$$

IND-CCA2 security. A cryptosystem is indistinguishable under an adaptive chosen cyphertext attack if every probabilistic polynomial time adversary has only a negligible advantage over random guessing, i.e it wins the above game with a probability $1/2+\epsilon(k)$, where $\epsilon(k)$ is a negligible function in the security parameter k. Although the adversary knows x_0, x_1 and pke the probabilistic nature of the encryption function means that the encryption of x_b will be only one of many valid ciphertexts, and therefore encrypting x_0, x_1, and comparing the resulting ciphertexts with the challenge ciphertext does not afford any advantage to the adversary. The game $\mathbb{G}^{\mathsf{ind}-\mathsf{cca2}}$ behaves as follows:

1. The challenger $\mathcal{C}_f^{\mathsf{ind}-\mathsf{cca2}}$ chooses a key pair $(\mathsf{pke}, \mathsf{ske})$ and returns pke.
2. The adversary $\mathcal{A}$ may perform q_1 number of encryption oracle or decryption oracle to the challenger.
3. $\mathcal{A}$ submits two distinct chosen messages x_0 and x_1 and sends them to $\mathcal{C}_f^{\mathsf{ind}-\mathsf{cca2}}$.
4. The challenger uniformly chooses a bit $b \in \{0,1\}$, and returns $f(\mathsf{pke}, x_b)$ to the adversary.
5. Upon receiving $C = f(\mathsf{pke}, x_b)$, the adversary may make further calls to the decryption oracle, but may not submit the challenge ciphertext C.
6. Finally, the adversary halts and outputs a guess d of the bit b used by the challenger.

Definition 14. *([General security of ind-cca2-scheme.]) A scheme f is considered as* $\mathsf{ind} - \mathsf{cca2}$*-secure by indistinguishability under an adaptative chosen ciphertext attack if any adversary $\mathcal{A}$ running in time t and making at most q queries to its challenger $\mathcal{C}_f^{\mathsf{ind}-\mathsf{cca2}}$, given an encryption of a message randomly chosen from a two-element message space, cannot distinguish efficiently the encryption of one of both messages. We can evaluate the* $\mathsf{ind} - \mathsf{cca2}$*-advantage of a such adversary, denoted $\mathsf{Adv}_f^{\mathsf{ind}-\mathsf{cca2}}(\mathcal{A})$:*

$$\mathsf{Adv}(\mathcal{A}_f^{\mathsf{ind}-\mathsf{cca2}})) = Pr[\mathsf{Adv}(\mathcal{A}_f^{\mathsf{ind}-\mathsf{cca2}}) \to 1 | b \leftarrow \{0,1\}, \\ (\mathsf{pke}, \mathsf{ske}) \leftarrow \{0,1\}^\kappa, (x_0, x_1) \in \{0,1\}^d \times \{0,1\}^d, \\ \mathcal{C} \to f(\mathsf{pke}, x_b)].$$

AE security. An AEAD-scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is ae-secure if every probabilistic polynomial time adversary has only a negligible advantage of the following indistinguishability game using the all-in-one formulation from [**?**]:

1. The challenger $\mathcal{C}_{\Pi}^{\mathsf{ae}}$ chooses a key $K \leftarrow \mathcal{K}$ and a bit $b \in \{0,1\}$. If $b = 0$, it uses the encryption and decryption functions of the model Real_{Π}. Otherwise, it uses the functions of the model Ideal_{Π}. These models are described in the figure 9.
2. The adversary $\mathcal{A}$ may perform q queries $\mathcal{E}$ and $\mathcal{D}$ such as for any tuples (N, A, M, A', M'), if the query $C = \mathcal{E}(K, N, A, M)$ has been required, then the two queries $\mathcal{E}(K, N, A', M')$ and $\mathcal{D}(N, A, C)$ cannot be required.
3. The adversary $\mathcal{A}$ outputs its guess d of the bit b used by the challenger.

Definition 15. *([General security of ae-scheme.]) An AEAD-scheme Π is a triple of algorithms $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ where $\mathcal{K}$ is a randomized algorithm that samples a key $K \in \{0,1\}^*$, $\mathcal{E}$ is a deterministic algorithm that maps the key K, a nonce N, additional data A, and a message M to a ciphertext C, and $\mathcal{D}$ is a deterministic algorithm that (K, N, A, C) to M. We assume that for all $K, N, A, M \in \{0,1\}^*$, we have $\mathcal{D}(K, N, A, \mathcal{E}(K, N, A, M)) = M$. We consider that a such scheme is* ae*-secure if any adversary $\mathcal{A}$ running in time t and making at most q queries to its challenger $\mathcal{C}_{\Pi}^{\mathsf{ae}}$ cannot success with a non-negligible advantage. A such advantage, denoted $\mathsf{Adv}_{\Pi}^{\mathsf{ae}}(\mathcal{A})$, is defined as follows:*

$$\mathsf{Adv}_{\Pi}^{\mathsf{ae}}(\mathcal{A}) = \left| \Pr(\mathcal{A} \to 1 | \mathsf{Real}_{\Pi}) - \Pr(\mathcal{A} \to 1 | \mathsf{Ideal}_{\Pi}) \right|.$$

B Evaluation

In proposing our variant of AKA we explicitly or implicitly addressed several attacks. We discuss these below, referring the reader to Figure 10 for a better overview.

Server Corruptions : The original AKA protocol only offers a degree of key-indistinguishability and impersonation security, only in the absence of server corruptions. Since servers are trusted to run the authenticated key-exchange step, corrupting a server compromises any security of a channel this server establishes; however, in the AKA routine, this flaw is exacerbated, since the corruption results can be re-used later in non-vulnerable areas. This is an active, and rather complex attack, but it is highly parallelizable and has a great security impact. To mitigate this risk, we added a server-specific, unique, publicly-known identifier $\mathsf{Res_S}$, which is now given as input to all the cryptographic functions.

Client Confidentiality : IMSI catcher attacks compromise the client's identity in a direct way (the adversary learns a static identifier). This attack can be run (with a reduced success probability) even by passive attackers, and it is highly parallelizable. The consequence is that multiple clients can be tracked simultaneously in a mass-surveillance operation. We mitigate such risks by ensuring that no static identifier is leaked through, by using a PKE scheme, in which only the operators have secret and public keys.

Client Unlinkability : Even if the adversary cannot track a user back to a permanent identifier, she can still try to distinguish between two chosen users, e.g. by causing some unusual protocol steps. Attacks like distinguishing between two different failure messages (which are actively triggered by the adversary), injecting a message and then seeing its effect in a protocol run (which is accepted if the chosen client is compatible with the injected message, and rejected otherwise), or distinguishing between messages of distinct lengths allow client linkability. While not as versatile, nor as parallelizable as client confidentiality attacks, these threat nevertheless allow an insidious adversary to track a user that is singled-out for mass surveillance. In our variant, we make protocol executions for different users indistinguishable from one another, at the cost of larger TMSI values, a new index variable, using IND-CCA PKE encryption, and making the operator intervene in the case of an error.

Denial of Service : Apart from being a mean of breaking client-unlinkability, DoS attacks can also facilitate IMSI catchers, and add to the complexity of the AKA procedure. One way of causing a DoS in the original protocol is to send a random string as a replacement for the TMSI reallocation message. The client will parse this as a different TMSI than the intended one, and thus the server will need to request the user's IMSI in clear. We mitigate DoS attacks by using authenticated encryption for the TMSI reallocation and ensuring that no desynchronizations can occur.

Itinerary tracking : One disadvantage of AKA is that the client's past location is revealed during the protocol, allowing to track up to 1 user per LAI at any one time. We bypass this difficulty by only using current LAI values.

C Full protocol description

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

The basic framework is a challenge-response stateful protocol between two main actors: the HLR (Home Network Register) and the ME/USIM (Mobile Equipment/User Subscriber Identity Module). This protocol needs an intermediate entity, the VLR (Visited Network Register), as specified in Section 3.2. Both the ME/USIM and the HLR keep track of counters, denoted respectively $\mathsf{Sqn}_{\mathsf{C}}$ and $\mathsf{Sqn}_{\mathsf{HLR}}$; these sequence numbers are meant to provide entropy and enable network authentication (from HLR to ME/USIM). Technically, one can view the user's sequence number as an increasing counter, while the latter keeps track of the highest authenticated counter the user has accepted.

The AKA protocol uses a set of seven functions: $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$, $\mathcal{F}_4$, $\mathcal{F}_5$, $\mathcal{F}_1^*$, $\mathcal{F}_5^*$. The first two are used to authenticate a MAC answer, proving that both participants know the same subscriber key sk_{C} and the same operator key $\mathsf{sk}_{\mathsf{op}}$. 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 $\mathsf{sk}_{\mathsf{op}}$.

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 $\mathsf{sk}_{\mathsf{op}}$, 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 home network, using the $\mathcal{F}_1^*$ and $\mathcal{F}_5^*$ to authenticate this value much in the same way that the server has authenticated its own sequence number and random value. Figure 11 details the challenge-response of AKA procedure.

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

Identification. Globally, the procedure starts with the identification of the ME/USIM to VLR when the user equipment switches on. At first, the mobile equipment receives a user request from the VLR and then responds in cleartext, with a temporary identity, called TMSI, which is known by the ME/USIM and VLR. A TMSIis a local number and its construction is specified in the technical specification 23.003 [ref]. This value can be used only in a specific given area: the TMSI is always accompanied by the Location Area Identification (LAI) to avoid ambiguities. The VLR (globally the network) manages suitable data based which keeps the relation between the IMSIs and the TMSIs. A new TMSI must be allocated at least in each location updating procedure, i.e when you use a TMSI we need to replace it by a fresh value. So the mobile station de-allocates the old value and allocates the fresh temporary identity from the VLR. We note that this value need to be store in a non volatile memory with its LAI in order to they are not lost when the ME/USIM switches off.

When the ME/USIM receives a *user identity request* from the VLR, it sends its $\mathsf{TMSI_o}\|\mathsf{LAI}$. When it receives its value, the ME/USIM verifies if the LAI matches the current ME/USIM. If it is not the case, it starts a *Local* TMSI *Unknown Procedure*. Otherwise, it tries to recover the corresponding permanent identity using its suitable data based to accept the identification. If a such value cannot be recover, it sends a *Permanent Identity Request* to the ME/USIM which answers with its IMSI. All these flows are exchanged in cleartext.

Local TMSI Unknown Procedure: As we said previously, if the LAI does not match the VLR, a *Local TMSI Unknown Procedure* has been engaged. Globally, it askes to the previous VLR_0 the relation with its TMSI. Then, either its received the corresponding IMSI or a "error message" implying a *Permanent Identity Request* to the mobile station.

Permanent Identity Request: In some cases, notably when the user cannot be identified with any temporary identity, the identification of a user on the radio path by means of the permanent subscriber identity, called IMSI. This basic procedure is sum up in the following figure:

The request and its answer are sent in cleartext. The procedure is used too when the user registers for the first time in a serving network.

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 [3]. The authentication vector is generated as follows:

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

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

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

The ME/USIM verifies the freshness of the received authentication token. To this end, it recovers the sequence number by computing the anonymity key AK which in its own turn depends on three values: $\mathsf{sk_C}$, $\mathsf{sk_{op}}$, and the received R. Then, the user verifies the received $\mathsf{Mac_S}$ computing $\mathcal{F}_1(\mathsf{sk_C}$, $\mathsf{sk_{op}}$, R, Sqn , AMF) with the received value R and the Sqn. If they are different, the user sends *authentication failure* message back to the VLR and the user abandons the procedure. In case the execution is not aborted, the ME/USIM verifies if the received Sqn value is in a correct range relatively to a stored value $\mathsf{Sqn_C}$[10]. 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 14.

The $\mathsf{Mac_S}$ value does not only ensure integrity, but also the authentication of the network by ME/USIM. If the two previous verifications are successful i.e if the received authentication token is fresh, the network

[10] 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.

is authenticated by the ME/USIM. Then, the ME/USIM computes CK, IK and 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 Res $=$ $\mathsf{Mac_C}$, the VLR successfully authenticates the ME/USIM. Otherwise, the VLR will initiate an *authentication failure* report procedure with the HLR. Note that the verification of the sequence number by the ME/USIM will cause the rejection of any attempt to re-use an authentication token more than once.

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

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

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

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

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

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

More precisely, the HLR retrieves the $\mathsf{Sqn_C}$ by computing $\mathcal{F}_5^*(\mathsf{sk_C}, \mathsf{sk_{op}}, \mathsf{R}) \oplus \lfloor \mathsf{Auts} \rfloor_{48}$. Then, it verifies if the incremented $\mathsf{Sqn_{HLR}}$ is in the correct range relatively to $\mathsf{Sqn_C}$. If the $\mathsf{Sqn_{HLR}}$ verifies this property, it sends a new list of authentication data vectors initiated with $\mathsf{Sqn_{HLR}}$ else HLR verifies the value of Mac^*If this step is successful, it resets the value of $\mathsf{Sqn_{HLR}}$ to $\mathsf{Sqn_{HLR}} := \mathsf{Sqn_C}$ and sends a new list of authentication data vectors initiated with this updated $\mathsf{Sqn_{HLR}}$. This list may also contain only a single authentication vector. Figure 14 details this re-synchronization procedure.

Re-allocation of the TMSI**.** At the end of the key derivation, the both entities (ME/USIM and VLR) need to be update with a fresh value. To allocate a fresh value, denoted TMSI_n, the VLR generates this value in the same LAI [11]. Only after a successful identification based on the old TMSI_o and a successful key setting permitting to share the ciphering key CK (by the AKA protocol). The VLR forges the new TMSI_n and sends it to the ME/USIM in a ciphered mode by A5/3 algorithm (globally an encryption with KASUMI more details in annex C and section 4.2 in TS 43.020) based the derived key CK. The ME/USIM recovers with the key CK the new TMSI_n and stored it and de-allocate TMSI_o. Then the ME/USIM sends an "acknowledge message" in cleartext to prevent its allocation. After receiving this message, the VLR de-allocates the TMSI_o and stores TMSI_n. If the VLR does not receive a such message, the network shall maintain the association between the old TMSI and the IMSI and between the new TMSI and the IMSI. For the next identification, the mobile station can used the both TMSI (TMSI_o and TMSI_n). This will allow the network to determine the TMSI stored in the ME/USIM; the association between the other TMSI and the IMSI shall then be deleted, to allow the unused TMSI to be allocated to another ME/USIM.

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).

[11] They are not recommended methods to generate the TMSI as the spec 23.003 (*"the structure and coding of it can be chosen by agreement between operator and manufacter in order to meet local needs"*). They provide only one advice (some parts of the TMSI may be related to the time)

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 $\mathsf{ME/USIM}$.
- Five 128-bit constants c_1,c_2,c_3,c_4,c_5 which are Xored onto intermediate variables and are defined as follows:
  - $\mathsf{c}_1[i] = 0, \forall i \in \{0, 127\}$.
  - $\mathsf{c}_2[i] = 0, \forall i \in \{0, 127\}$, except that $\mathsf{c}_2[127] = 1$.
  - $\mathsf{c}_3[i] = 0, \forall i \in \{0, 127\}$, except that $\mathsf{c}_3[126] = 1$.
  - $\mathsf{c}_4[i] = 0, \forall i \in \{0, 127\}$, except that $\mathsf{c}_4[125] = 1$.
  - $\mathsf{c}_5[i] = 0, \forall i \in \{0, 127\}$, except that $\mathsf{c}_5[124] = 1$.
- Five integers r_1,r_2,r_3,r_4,r_5 in the range $\{0, 127\}$ which define amounts by which intermediate variables are cyclically rotated and are defined as follows: $\mathsf{r}_1 = 64; \mathsf{r}_2 = 0; \mathsf{r}_3 = 32; \mathsf{r}_4 = 64; \mathsf{r}_5 = 96$.

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

$$\mathsf{Top}_C = \mathsf{sk_{op}} \oplus \mathsf{AES}_{\mathsf{sk_c}}(\mathsf{sk_{op}})$$

. 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 15

TUAK algorithms:

TUAK [2] is a set of algorithms which aims to achieve secure mutual authentication and key generation properties. The TUAK algorithms are based on the TUAK permutation, which in turn relies on a truncation of the internal permutation function of Keccak. Moreover, for efficiency reasons, only one or two iterations of the internal TUAK permutation, $\mathsf{f}_{\mathsf{Keccak}}[1600]$, is used.

The functions $\mathcal{F}_1$ (respectively $\mathcal{F}_1^*$) and $\mathcal{F}_2$ must provide authentication while the functions $\mathcal{F}_3$, $\mathcal{F}_4$ and $\mathcal{F}_5$ (respectively $\mathcal{F}_5^*$) 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 256-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.
– A subscriber key $\mathsf{sk_C}$ shared out of band between the HLR and $\mathsf{ME/USIM}$ 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 $\mathsf{Inst'}$ are fixed binary variables of 8 bits, specified in [2] and different depends 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 at 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 HLR 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. Next only the first 64 bits are used to compute the $\mathsf{Mac_S}$ value. To generate derived keys and the response of $\mathcal{F}_2$, one initializes a second state for this function, too, namely,

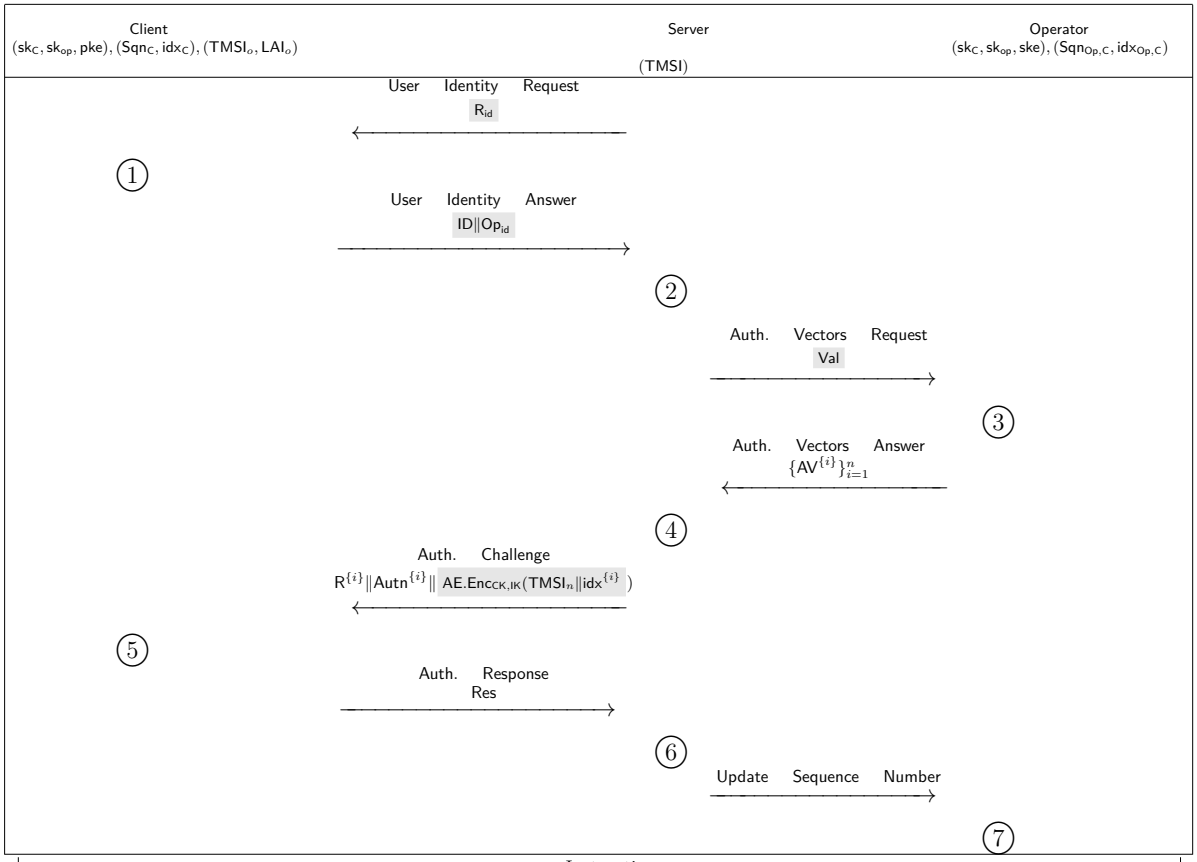
$$\mathsf{Val}_2 = \mathsf{Top}_C\|\mathsf{Inst'}\|\mathsf{AN}\|\mathsf{R}\|0^{64}\|\mathsf{Key}\|\mathsf{Pad}\|1\|0^{512}.$$

Then, the TUAK permutation is applied on Val_2 and one obtains the value Out. Finally, one derives the response $\mathsf{Mac_C}$ and the derived keys from the resulting Out:

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

This is also depicted in Figure 16.

The way the output of the functions is truncated and used is the reason why TUAK is called a *multi-output function*. This notion is one of the main differences with $\mathsf{MILENAGE}$ and has a no-negligible impact on the efficiency of TUAK, as it saves a few calls of the internal function. However, this multi-output function property can be an issue for the security of the master key. Indeed, during one session we can have four calls of the same function with similar inputs but with a different truncation. Having different chunks of the same global 1600-state (called Out in our description)) can permit to recover the long-term key $\mathsf{sk_C}$ by the reversibility of the TUAK permutation. The union of all the different chunks provided during one session, gives at most only 432 bits on the 1600 bits. Thus, having multiple ooutputs may be hazardous in general, the Keccak based construction of TUAK allows this without compromising the long-term parameters.



Instructions:

Client	Server	Operator
①: Compute the identifier: If $\mathsf{flag_{TMSI}} := 0$ then $\mathsf{ID} = \mathsf{TMSI}$. Else, $\mathsf{ID} = \mathsf{PKE.Enc_{pke}}(f_5(\mathsf{keys}, \mathsf{R_{id}}, \mathsf{IMSI}, \mathsf{idx_C})\|\mathsf{R_{id}}\|\mathsf{IMSI}\|\mathsf{idx_C})$. $\mathsf{flag_{TMSI}} := 1$. ⑤: Compute AK using $\mathsf{R}^{\{i\}}$. Recover $\mathsf{Sqn}^{\{i\}}$ (from AK). Check $\mathsf{Mac_S}$ value. Compute: IK, CK; Retrieve the received index and the new TMSI. If abort caused or the AE does not verify, set $\mathsf{flag_{TMSI}} := 1$ and increment: $\mathsf{idx_C} := \mathsf{idx_C} + 1$. Else, check validity of $\mathsf{Sqn}^{\{i\}}$, i.e if one of the following conditions is correct: – $\mathsf{Sqn_C} = \mathsf{Sqn}^{\{i\}}$. – $\mathsf{Sqn_C} = \mathsf{inc}(\mathsf{Sqn}^{\{i\}})$ and $\mathsf{idx}^{\{i\}} = \mathsf{idx_C} + 1$. If the first condition is accepted: reset the index $\mathsf{idx_C}$, update the sequence number $\mathsf{Sqn_C} = \mathsf{inc}(\mathsf{Sqn_C})$. If the second condition is accepted: $\mathsf{idx_C}$=$\mathsf{idx_C}$+1. Compute $\mathsf{Res} := \mathcal{F}_1^*(\mathsf{keys}, \mathsf{R}^{\{i\}}, \mathsf{Sqn}^{\{i\}}, \mathsf{Res_S}, \mathsf{AMF})$. Update the internal index. Allocate the new TMSI. $\mathsf{flag_{TMSI}} := 0$.	②: Process the identifier ID: If the identifier is a TMSI then Val = IMSI. Otherwise, $\mathsf{Val} = (\mathsf{ID}, \mathsf{R_{id}})$. ④: Store $\{\mathsf{AV}^{\{i\}}\}_{i=1}^{n}$. Choose $\mathsf{AV}^{\{i\}}$ one by one in order. Then, it sends the authentication challenge and the new couple $(\mathsf{TMSI}_n, \mathsf{idx}^{\{i\}})$ encrypted and authenticated by the session keys. ⑥: If the authentication of the client is verified ($\mathsf{Res} \stackrel{?}{=} \mathsf{Mac_C}$), then they ask to the server the update of its sequence number. Otherwise, the protocol is aborted.	③: Verify the identity of the client with Val. If this holds, retrieve $\mathsf{idx_C}$, set $\mathsf{idx_{Op,C}} := \mathsf{idx_C}$ Generate $(\mathsf{R}^{\{1\}}, ..., \mathsf{R}^{\{n\}})$. Denote: $\mathsf{keys} := (\mathsf{sk_C}, \mathsf{sk_{op}})$. For each $i = 1, \ldots, n$, compute: $\mathsf{Mac_S} \leftarrow \mathcal{F}_1(\mathsf{keys}, \mathsf{R}^{\{i\}}, \mathsf{Sqn}^{\{i\}}, \mathsf{Res_S}, \mathsf{AMF})$, $\mathsf{Mac_C} \leftarrow \mathcal{F}_1^*(\mathsf{keys}, \mathsf{R}^{\{i\}}, \mathsf{Sqn}^{\{i\}}, \mathsf{Res_S}, \mathsf{AMF})$, $\mathsf{CK} \leftarrow \mathcal{F}_3(\mathsf{keys}, \mathsf{R}^{\{i\}}, \mathsf{Sqn}^{\{i\}}, \mathsf{Res_S}, \mathsf{AMF})$, $\mathsf{IK} \leftarrow \mathcal{F}_4(\mathsf{keys}, \mathsf{R}^{\{i\}}, \mathsf{Sqn}^{\{i\}}, \mathsf{Res_S}, \mathsf{AMF})$, $\mathsf{AK} \leftarrow \mathcal{F}_5(\mathsf{keys}, \mathsf{R}^{\{i\}}, \mathsf{Res_S})$, $\mathsf{Autn}^{\{i\}} \leftarrow (\mathsf{Sqn}^{\{i\}} \oplus \mathsf{AK})\|\mathsf{AMF}\|\mathsf{Mac_S}$, $\mathsf{Sqn}^{\{i\}} \leftarrow \mathsf{inc}(\mathsf{Sqn}^{\{i-1\}})$, $\mathsf{AV}^{\{i\}} := (\mathsf{R}^{\{i\}}, \mathsf{CK}, \mathsf{IK}, \mathsf{Autn}^{\{i\}}, \mathsf{Mac_C}, \mathsf{idx}^{\{i\}})$, with $\mathsf{Sqn}^{\{1\}} := \mathsf{Sqn_{Op,C}}$, $\mathsf{idx}^{\{1\}} := \mathsf{idx_{Op,C}}$, $\forall i \neq 1, \mathsf{idx}^{\{i\}} = 0$. End for. ⑦ : Update the sequence number: $\mathsf{Sqn_{Op,C}} \leftarrow \mathsf{inc}(\mathsf{Sqn_{Op,C}})$. Reset the index $\mathsf{idx_{Op,C}}$.

Fig. 5. Our fixed AKA Procedure.

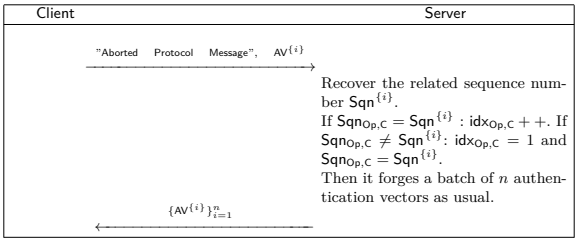


Fig. 6. Procedure after an abort.

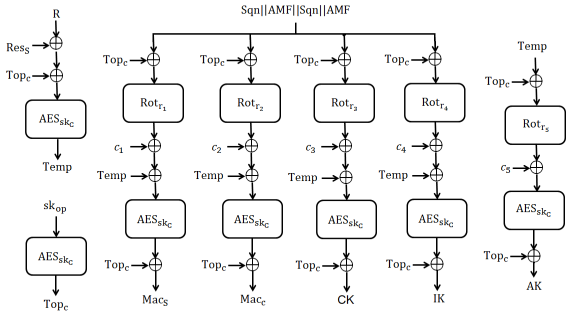


Fig. 7. Updated Milenage.

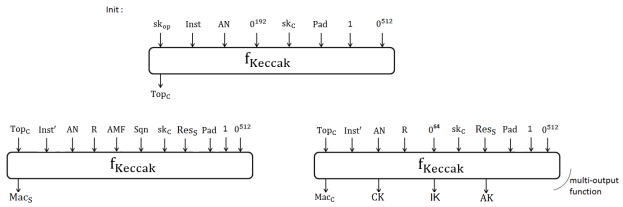


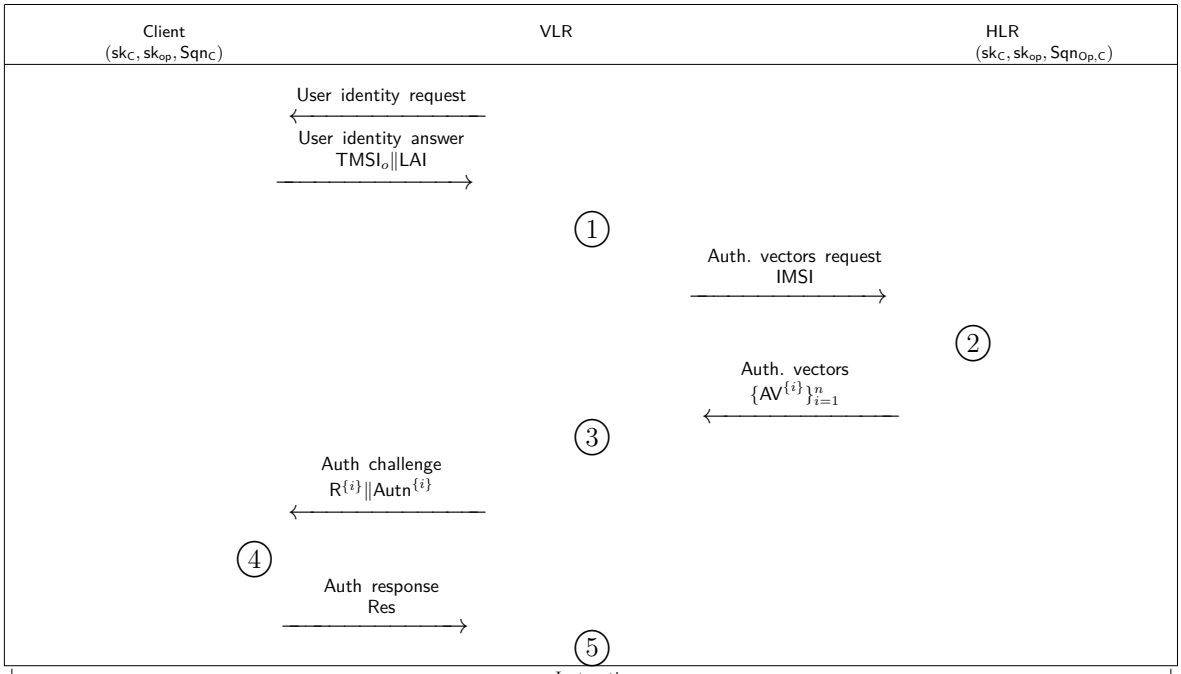
Fig. 8. Updated TUAK

$\mathbf{Real}_{\mathbf{\Pi}}$	$\mathbf{Ideal}_{\mathbf{\Pi}}$
Initialization	Initialization
$K \leftarrow \mathcal{K}$	$K \leftarrow \mathcal{K}$
Oracle AE.Enc	Oracle AE.Enc
Input: $(N, A, M) \in (\mathcal{N}, \mathcal{A}, \mathcal{M})$ $C \leftarrow \mathcal{E}(K, N, A, M)$ return C	Input: $(N, A, M) \in (\mathcal{N}, \mathcal{A}, \mathcal{M})$ $C' \leftarrow \mathcal{E}(K, N, A, M)$ $C \leftarrow \{0,1\}^{\lvert C' \rvert}$ return C
Oracle AE.Dec	Oracle AE.Dec
Input: $(N, A, C) \in (\mathcal{N}, \mathcal{A}, \mathcal{C})$ $M \leftarrow \mathcal{D}(K, N, A, C)$ return M	Input: $(N, A, C) \in (\mathcal{N}, \mathcal{A}, \mathcal{C})$ return $\perp$

Fig. 9. Real and Ideal security game for AEAD-schemes.

Added countermeasures	Cost	Attacks it Prevents	Attack Impact
Client sends encrypted IMSI	- Needs IND-CCA PKE encryption - Simple PKI (only operators)	Client Confidentiality: (IMSI Catchers)	Trace many users - Parallelizable - Passive/Active
	Large TMSI size	Client unlinkability: Distinguish TMSI/IMSI msg.	Trace 1 user: - Non-parallelizable - Active only
Authenticate TMSI reallocation (see also: index)	New reallocation alg.	Client unlinkability: (Denial-of-Service)	Trace many users - Parallelizable - Active only
		Client unlinkability: Distinguish TMSI/IMSI	Trace 1 user - Non-parallelizable - Active only
Index idx_C, idx_S	New 1-bit state variable	Client unlinkability: Prompt resynch, distinguish	Trace 1 user - Non-parallelizable - Active only
		S.Imp-resistance: Challege is un-replayable	Impersonate servers - Parallelizable - Active only
Introducing Res_S	- New server identifier - Changed crypto algs.	S.Imp-resistance k.ind-security S.sound-security (Server Corruptions)	Break sec. channel - Parallelizable - Needs corruptions - Active only
Use only current LAI	- Clients must know LAI - Clients store Res_S	Location privacy: (Track past LAI)	Trace 1 user per LAI - Non-Parallelizable - Passive

Fig. 10. Assessment of our AKA variant: cost and effect of coutermeasures.



Instructions:

Client	VLR	HLR
④: 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: $\mathsf{CK} \leftarrow \mathcal{F}_3(\mathsf{sk_C}, \mathsf{sk_{op}}, \mathsf{R}^{\{i\}}, \mathsf{Sqn}^{\{i\}}, \mathsf{AMF})$, $\mathsf{IK} \leftarrow \mathcal{F}_4(\mathsf{sk_C}, \mathsf{sk_{op}}, \mathsf{R}^{\{i\}}, \mathsf{Sqn}^{\{i\}}, \mathsf{AMF})$, 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*	①: Verify if the LAI matches the VLR. Otherwise *Local TMSI Unknown Procedure*. Then, if it retrieves the IMSI corresponding to the $TMSI_o$. Otherwise, it sends a *Permanent Identity Request.* ③: Store $\{\mathsf{AV}^{\{i\}}\}_{i=1}^{n}$. Choose $\mathsf{AV}^{\{i\}}$ one by one in order. Then it sends the related challenge. ⑤: $\mathsf{Res} \stackrel{?}{=} \mathsf{Mac_C}$.	②: 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. 11. The AKA Procedure.

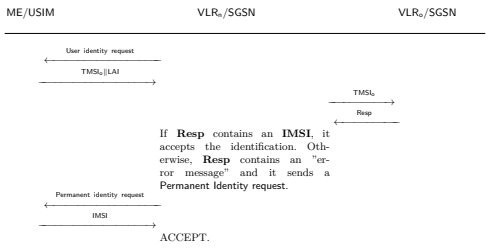


Fig. 12. Local TMSI Unknown Procedure

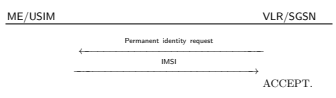
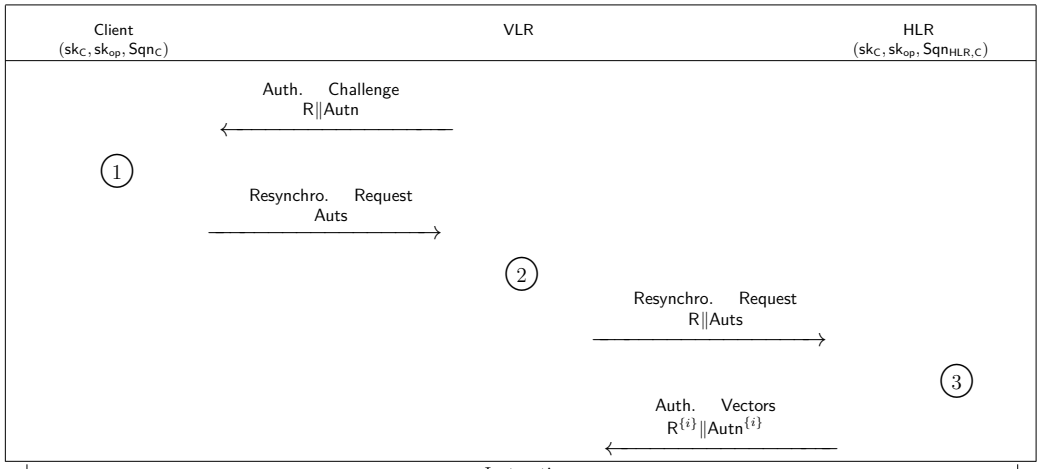


Fig. 13. Identification by the permanent identity.



Instructions:

Client	VLR	HLR
①: Compute the value AK with the value R. Recover Sqn with AK. Check if the received $\mathsf{Mac_S}$ is correct. Check if Sqn is in the correct range related to the stored $\mathsf{Sqn_C}$. Re-synchronization Procedure: Compute $\mathsf{AK}^* \leftarrow \mathcal{F}_5^*(\mathsf{sk_{op}}, \mathsf{sk_C}, \mathsf{R})$ and $\mathsf{Mac}^* \leftarrow \mathcal{F}_1^*(\mathsf{sk_{op}}, \mathsf{sk_C}, \mathsf{R}, \mathsf{Sqn_C}, \mathsf{AMF})$. Forge $\mathsf{Auts} = (\mathsf{Sqn_C} \oplus \mathsf{AK}^*)||\mathsf{Mac}^*)$.	②: Add the R value.	③: Compute the value AK^* with the value R. Recover $\mathsf{Sqn_C}$ with AK^*. Check if the incremented $\mathsf{Sqn_{HLR,C}}$ is in the correct range related to $\mathsf{Sqn_C}$. If it is not the case and if only the received Mac^* is correct, then $\mathsf{Sqn_{HLR,C}} \leftarrow \mathsf{Sqn_C}$. Otherwise, it aborts the procedure. It sends a new list of authentication data vectors initiated with $\mathsf{Sqn_{HLR,C}}$.

Fig. 14. The re-synchronization procedure of AKA protocol.

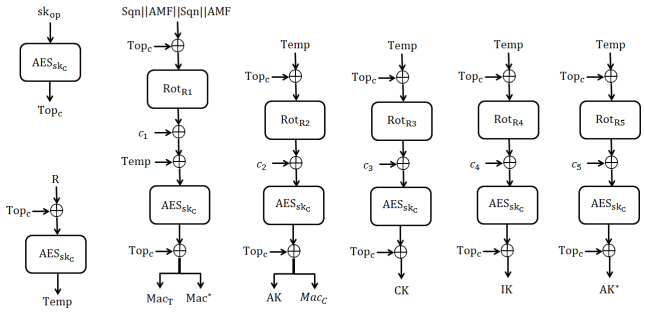


Fig. 15. MILENAGE diagram.

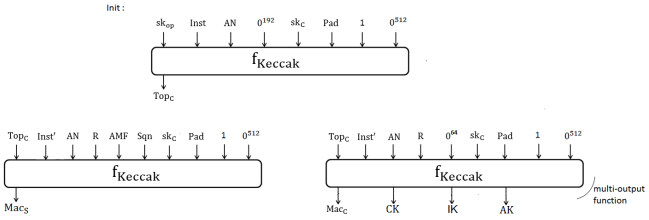


Fig. 16. TUAK diagram.