Akman, Gizem; Ginzboorg, Philip; Damir, Mohamed Taoufiq; Niemi, Valtteri

# Privacy-Enhanced AKMA for Multi-Access Edge Computing Mobility

*Published in:*
Computers

*DOI:*
10.3390/computers12010002

Published: 01/01/2023

*Document Version*
Publisher's PDF, also known as Version of record

computers

MDPI

*Article*

# Privacy-Enhanced AKMA for Multi-Access Edge Computing Mobility †

**Gizem Akman [1,2,*], Philip Ginzboorg [3], Mohamed Taoufiq Damir [1,2] and Valtteri Niemi [1,2]**

[1] Department of Computer Science, University of Helsinki, Pietari Kalmin katu 5, 00560 Helsinki, Finland
[2] Helsinki Institute for Information Technology (HIIT), Konemiehentie 2, 02150 Espoo, Finland
[3] Huawei Technologies Oy, Itämerenkatu 9, 00180 Helsinki, Finland
* Correspondence: gizem.akman@helsinki.fi
† Computational Science and Its Applications (ICCSA 2022)-2nd Workshop on Privacy in the Cloud/Edge/IoT World (PCEIoT 2022).

**Abstract:** Multi-access edge computing (MEC) is an emerging technology of 5G that brings cloud computing benefits closer to the user. The current specifications of MEC describe the connectivity of mobile users and the MEC host, but they have issues with application-level security and privacy. We consider how to provide secure and privacy-preserving communication channels between a mobile user and a MEC application in the non-roaming case. It includes protocols for registration of the user to the main server of the MEC application, renewal of the shared key, and usage of the MEC application in the MEC host when the user is stationary or mobile. For these protocols, we designed a privacy-enhanced version of the 5G authentication and key management for applications (AKMA) service. We formally verified the current specification of AKMA using ProVerif and found a new spoofing attack as well as other security and privacy vulnerabilities. Then we propose a fix against the spoofing attack. The privacy-enhanced AKMA is designed considering these shortcomings. We formally verified the privacy-enhanced AKMA and adapted it to our solution.

**Keywords:** MEC; 5G; AKMA; MEC mobility; security; privacy; ProVerif; formal verification



**Citation:** Akman, G.; Ginzboorg, P.; Damir, M.T.; Niemi, V. Privacy-Enhanced AKMA for Multi-Access Edge Computing Mobility. *Computers* **2023**, *12*, 2. https://doi.org/10.3390/computers12010002

Academic Editor: Paolo Bellavista

Received: 15 November 2022
Revised: 9 December 2022
Accepted: 15 December 2022
Published: 20 December 2022



## 1. Introduction

The fifth generation (5G) mobile network is developed to have reliable, energy-efficient, and high-capacity service with low latency [1]. Ultimately, the 5G is expected to have data rates up to 10 Gbps, service level latency below 1 ms, ubiquitous connectivity of 100%, improved battery life, and support for 300,000 devices within a single cell [1–3].

The Internet of Things (IoT) is a growing ecosystem with a massive number of interconnections between heterogeneous physical objects. The number of IoT devices is increasing exponentially; hence these devices generate a larger and larger amount of data. However, IoT devices are resource-constrained. They typically have low battery power, low storage capacity, small processing power, and limited communication capacity [1,4]. Therefore, IoT applications require a decentralized local service that provides location awareness, reliability, low latency [1], and access to centralized cloud computing facilities for data processing and storage [4].

Cloud computing-based services support IoT applications in providing computing resources and storage. On the other hand, cloud computing-based services may have latency issues, jitter, service interruptions, and a single point of failure, which is inconvenient, especially for time-sensitive IoT services [1,3]. Multi-access edge computing (MEC) is developed to extend the capabilities of cloud computing to the edge of the mobile network and to provide improved quality of service (QoS) and quality of experience (QoE) [4,5]. In order to achieve this, MEC hosts are deployed to the edge of the mobile network.

MEC, together with 5G technology, provides proximity, low latency, high bandwidth, location awareness, real-time responsiveness, security, and privacy protection, proximate

data outsourcing, and mobility support [1,3,4]. Examples of IoT applications that can benefit from MEC, supported by 5G, include ultra-high definition (UHD) video streaming, augmented reality (AR), virtual reality (VR), mixed reality (MR), tactile internet, machine-type communication (MTC), machine-to-machine (M2M), unmanned aerial vehicle (UAV), vehicle-to-everything (V2E), intelligent transportation system(s) (ITS), serious gaming, factory automation, healthcare, smart grid, smart home, smart city, smart retail, and smart agriculture [1–3].

User equipment (UE) mobility –where a UE that is initially connected to the source MEC host is relocated to the target MEC host– has varying impacts on MEC applications. Some are not affected by UE mobility, e.g., when the user state does not need to be maintained for the service. On the other hand, some MEC applications are more sensitive to UE mobility. An example is when the service requires continuous connectivity between the device and the MEC application on the network side. This case requires a seamless handover from one MEC host to another [6]. In order to provide seamless relocation, it is necessary to transfer the user context (i.e., user-specific application state) between the source and target MEC hosts [7,8]. The transfer of the user context can happen in three ways: device application-assisted state transfer, MEC-assisted state transfer, and application self-controlled state transfer [7].

The MEC applications often process data related to users, share secret keys, and store private data. Therefore, relocating services between different MEC hosts may introduce security and privacy issues [9], and it is important to study how UE mobility could be handled while maintaining security and privacy. For example, it is necessary to run authentication with the target MEC application, and authorization should be provided before relocation. Otherwise, an unauthorized UE may start using the MEC application, or the UE may share its data with an unauthorized MEC application [10].

The need for secure authentication between the users and the application functions led to the development of generic bootstrapping architecture (GBA) in the third-generation partnership project (3GPP). The main purpose of GBA is to enable authentication and establish shared keys between the UE and network application function (NAF). In order to achieve this, the 3GPP authentication infrastructure is used [11,12]. Subscription of UE implies that the mobile network operator (MNO) and UE share a permanent key K. The UE is known by a unique identifier, which is called the international mobile subscriber identity (IMSI) in earlier networks and subscription permanent identifier (SUPI) in 5G. After the Authentication and Key Agreement (AKA) protocol is run successfully, MNO and UE also share session keys. Another key is derived from these shared session keys to be shared between UE and NAF [11,13]. The GBA was designed originally for 3G networks and later extended to 4G (or LTE) [13]. A new framework, authentication and key management for applications (AKMA), was developed as an alternative to GBA for 5G [14]. The AKMA framework also supports authentication over non-3GPP network access [13].

We consider a user who is a MEC application client and wants to use the MEC application in the MEC host. The problem we are addressing in this paper is how to secure the connection between the user and the MEC application, (i) when the user is registering to the main server of the MEC application, (ii) when the user connects to the MEC application in the MEC host, (iii) when the mobile user changes the MEC hosts while staying in the same mobile network.

In order to resolve the problem, we propose a secure and privacy-friendly mechanism that enables both static and mobile usage of a MEC application. The solution includes the following protocols: (5A) registration of the user to the main server of the application, (5B) updating the user information and shared keys with the main server of the application, (5C) usage of the application in the MEC host when the user is stationary, and (5D) usage of the application in several MEC hosts when the user is moving. The solution utilizes a privacy-enhanced version of the 3GPP AKMA service for authentication and key sharing.

We make the following contributions to this paper.

1. We propose a solution for static and mobile usage of a MEC application, as described above.
2. We formally verify AKMA using ProVerif [15].
3. We found a new spoofing attack, as well as several privacy vulnerabilities, in the current AKMA specification [14]. We also have a fix against the new attack, and we informed a 3GPP delegate about both the attack and its fix.
4. We propose a privacy-enhanced version of AKMA that fixes the above shortcomings.
5. We formally verify the privacy-enhanced AKMA using ProVerif.
6. We adopt the privacy-enhanced AKMA in the AKMA profiles for TLS and implement these profiles in the proposed protocols for the setting mentioned above.

Necessary background information and related work are provided in Section 2. The problem statement is described along with the system model in Section 3. Section 4 includes AKMA, the new attack against AKMA, and privacy-enhanced AKMA with formal verifications of both AKMA and PE-AKMA, done by ProVerif. The solution with four protocols is presented in Section 5. The paper ends with an analysis of the solution in Section 6 and the final remarks in Section 7. In addition, there are six appendices.

## 2. Review and Related Work

In this section, we provide background information and summarize related work.

### *2.1. MEC Framework*

European Telecommunications Standards Institute (ETSI) initiated MEC standardization in 2014 [16]. The framework of MEC is illustrated in Figure 1 and it consists of three levels [6]: MEC system, MEC host, and network.

The MEC system level includes the MEC orchestrator (MEO) and user application. MEO has an overview of the MEC system, selects a new host for relocation, and initiates the transfer from the source MEC host to the target MEC host [6,16].

The MEC host-level includes MEC host and MEC host-level management. The MEC host is deployed to the edge of the mobile network, consisting of a MEC platform, MEC applications, and virtualization infrastructure.

MEC functionality can be run over a cellular network, some other wide-area network, or, alternatively, over a local network such as Wi-Fi.

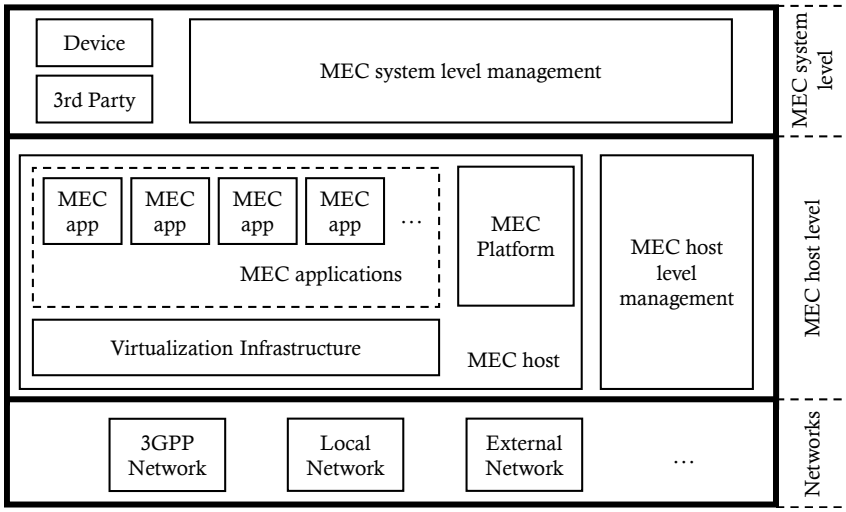


**Figure 1.** MEC framework, adapted from [6].

### *2.2. Authentication and Key Management for Applications (AKMA)*

Authentication and key management for applications (AKMA) is a key distribution service in the 5G system that supports authentication and security protection between the UE and applications. AKMA uses similar principles to those of generic bootstrapping

architecture (GBA) in legacy mobile networks. AKMA adapts to the architectural changes in the 5G system to meet the requirements for various scenarios, e.g., those related to the Internet of Things [17]. The authentication and key agreement rely on the credentials and identities of the mobile subscriber, and are pre-established with the mobile network operator (MNO) for 5G access [18]. A fundamental difference between AKMA and GBA is that AKMA bootstraps application security from the keys derived in the authentication and key agreement (AKA) during the network access authentication, whereas GBA is based on an AKA run that is independent of network access. More information about GBA is given in Appendix A.

It is important to notice that the AKMA alone does not yet provide authentication between UE and the application. Instead, AKMA is used with another protocol, e.g., TLS, which provides authentication between these two entities based on a shared key created by AKMA.

### *2.3. Formal Verification*

The formal verification of security protocols aim to model and check the security and privacy properties of cryptographic protocols and primitives with the proofs constructed in a fully mechanized manner [19]. The formal verification tries to prove or disprove the security properties. Since proving the security of a protocol is an undecidable problem, the termination of the proof process is not guaranteed [20]. Formal verification methods have been used to verify various mobile network protocols [21].

ProVerif and Tamarin are state-of-art tools for formal verification of the symbolic modeling and analysis of security protocols. Both use the Dolev–Yao adversary model [15,22]. Tamarin allows protocols to be modeled using expressive language based on multiset rewriting rules. These rules are then used to construct a labeled transition system with symbolic representations of the protocol state, messages, and knowledge of the attacker. The protocol state is composed of multiset facts that are a predicate applied to terms, and these facts represent the state information [20]. ProVerif uses the applied pi-calculus as a formal language, translates the protocol into a set of Horn clauses, then tries to conclude if some security property is falsified, i.e., finds an attack [21]. More information about ProVerif is provided in Appendix B.

### *2.4. Security and Authentication Properties*

In this section, we explain security properties that are important in our context and how they are implemented in ProVerif.

- **Secrecy** means that the attacker cannot reach the private information. Therefore, the secrecy queries verify the confidentiality and privacy of data. In ProVerif, the secrecy of the term M can be verified using the command `query attacker(M)`. If ProVerif returns `query not attacker (M) is true`, then the term M remains secret despite the presence of an attacker in the public channels [15].
- **Reachability** means that an event occurs in the protocol. In ProVerif, the reachability of event e can be checked with `query event(e)` [15].
- **Strong secrecy** means that the attacker cannot notice the differences between different sessions that follow from changing secrets in the protocol. It is similar to the concept of indistinguishability and semantic security in the computational proof-based approach in cryptography [15,19,23]. In ProVerif, strong secrecy for the term M is proved by the query `noninterf M` [15].
- **Weak secrets** are the secrets that are vulnerable to off-line guessing attacks, e.g., human-memorable passwords [15,23]. ProVerif tries to find proof that the attacker cannot distinguish a correct guess of the secret M from an incorrect guess with the query `weaksecret M` [15].
- **Forward secrecy** protects against an attacker that has a recording of some past (encrypted) sessions. More precisely, forward secrecy ensures that past recorded encrypted messages (session keys) will remain secret, despite the long-term keys being

compromised at some point. One can use ProVerif to show that even if some party in the protocol is corrupted (long-term keys are leaked), the secrets that are shared before the corruption are not revealed to the attacker. Assuming that a long-term key K leaked in the first execution (phase 0), forward secrecy for the subsequent protocol execution (phase 1) is checked by adding the process `phase 1; out(c,K)` to the main process, where c is a public channel [15].

Next, we describe the authentication properties as described in [24] and their implementation in ProVerif.

- The protocol guarantees agent A the **aliveness** of agent B, if whenever A completes a run of the protocol (apparently with B) then B has previously been running the protocol. Aliveness does not guarantee that B believes that B ran the protocol with A.
- The protocol guarantees agent A a **weak agreement** with agent B, if whenever A completes a run of the protocol (apparently with B) then B has previously been running the protocol, *apparently with A* [24].
- The protocol guarantees agent A a **non-injective agreement** with B, if whenever A completes a run of the protocol, apparently with B, then B has previously been running the protocol, apparently with A, *both A and B have an identical value for a data item M* [24]. The non-injective agreement does not rule out the case when A runs the protocol twice, and B takes part in only one of the runs. If the non-injective agreement is not satisfied, A can be subject to an impersonation attack [25].
- The protocol guarantees agent A an **injective agreement** with B if, whenever A completes a run of the protocol, apparently with B, then B has previously been running the protocol, apparently with A, both A and B agree on the message M, and *each such run of A corresponds to a unique run of B* [24]. If the injective agreement is not satisfied, A can be subject to a replay attack [25].

In ProVerif, for two events, e and e0, the query for `event(e(M))==>event(e0(N))` means that if an event e(M) has been executed, then event e0(N) has been previously executed. Thus, to prove authentication properties, the events are placed in the ProVerif script to capture authentication definitions. For example, we place an event at the end of agent A's process to capture the fact that "A completes the protocol". On the other hand, the claim "B has previously been running the protocol", can be modeled as "B has sent its last message or B has responded to a message (not necessarily to/from A)". ProVerif has built-in functionality to check injectivity according to the definition in [24], namely, `inj-event(e(M))==>inj-event(e0(N))` which additionally checks if the number of occurrences of e0(N) is greater than or equal to the number of occurrences of e(M).

It should be noted that the authentication properties are building on each other: the injective agreement implies a non-injective agreement, which implies a weak agreement, and which implies an aliveness.

### *2.5. Related Work*

Security plays an important role in the use cases, e.g., smart buildings, connected cars, healthcare [26], and unmanned aerial vehicle (UAV)-assisted MEC [27]. Examples of security and privacy threats are control takeover attacks that can lead to data manipulation, data loss, data recovery, and data leakage. In addition, malicious third-party MEC applications in the MEC host can cause serious security problems [26,28,29].

Khan et al. [30] propose a new privacy mode for AKMA, in which the Application Function (AF) and the mobile network communicate via the UE rather than directly. This mode implies bigger changes to the AKMA system than our privacy-enhanced AKMA, where the message paths are the same as in the current AKMA standard.

Kim et al. [31] propose a handover authentication protocol for distributed application functions in a 5G MEC environment using AKMA. In the proposed handover protocol, user context transfer is done via the local MEC AFs. They also provide formal verification of their protocol.

Yang et al. [32] conducted formal verification of AKMA with the Tamarin tool [22].

Niewolski et al. [33] propose a token-based authentication framework for the 5G MEC system. However, they do not consider the case where the user is mobile.

Several works address the security of mobile MEC users without putting a special focus on the privacy of the MEC user. Ali et al. [34] designed a token- or cookie-based authentication of mobile MEC users. In [35], the authors propose establishing an infrastructure of proxies to extend token-based authentication to multi-operator scenarios. Sanchez-Gomez et al. [36] use the pre-shared key extensible authentication protocol method (EAP-PSK) to authenticate the user and the MEC application.

Zhang et al. [37] designed a mobility support system (MSS) in the cellular network infrastructure and use that system to protect the privacy of mobile MEC users.

## 3. Statement of the Problem

The following scenario motivates our system model: Alice travels to another city in her home country. Assume she is in Finland and she travels from Helsinki to Tampere. On the way to Tampere, Alice uses a MEC application APPIFY, e.g., for video streaming, AR, or mobile gaming. Since Alice stays in her home country, it is very likely that MNO will not change during her trip. However, coverage of MEC is only local. Alice would need to change between several MEC hosts during the trip.

Nonetheless, she wants to continuously use APPIFY with the benefits of MEC services, such as low latency and high bandwidth. Additionally, she wants to protect her personal information from outsiders and other parties in the network. This information could include, e.g., the name of APPIFY, the content of her communication with APPIFY, and her identifiers. Alice also wants to protect her true identity and her identity in MNO from APPIFY.

The system model includes the following entities: the user equipment (UE), mobile network operator (MNO), MEC hosts, MEC applications, and the main server of the application. Figure 2 presents the settlement of these entities in the system model.

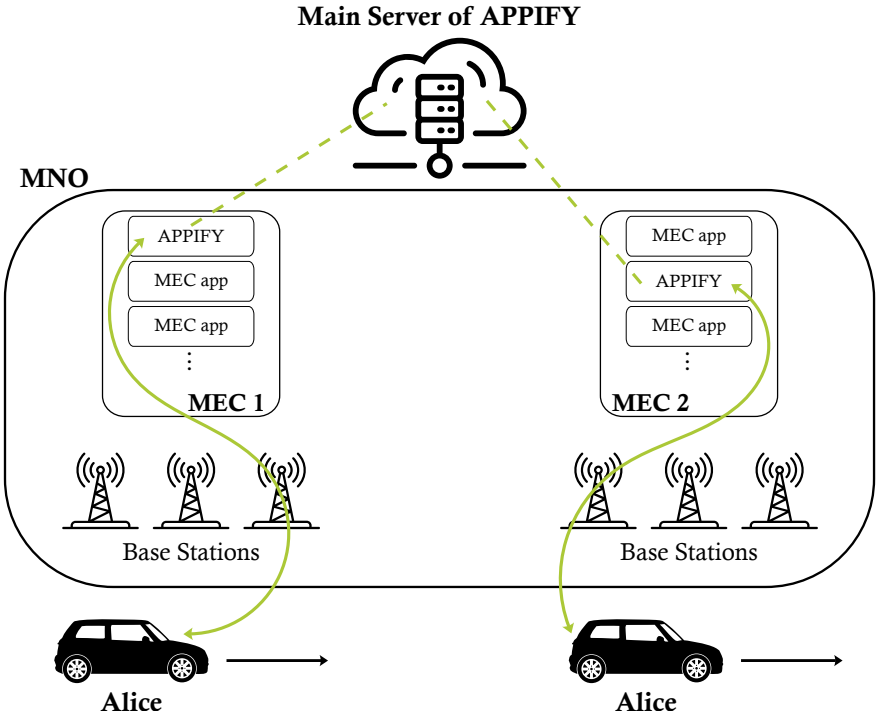


**Figure 2.** System model.

Alice is the user who possesses the user equipment UE, and she uses the application APPIFY. This application has a main server that can be reached via the internet. In addition

to this, instances of APPIFY servers reside in MEC hosts. APPIFY in each MEC host is synchronized with APPIFY main server. Alice connects to the APPIFY in the MEC host and obtains the service from there instead of the main server, hence enjoys the benefits of Multi-Access Edge Computing.

The MEC hosts are deployed in the network of MNO. There are only two MEC hosts in our illustration of the system model, presented in Figure 2, but there could be more. It should also be noted that there typically are several other MEC application servers in MEC hosts besides APPIFY.

As shown in Figure 2, Alice is on the move, and her connection is regularly transferred from one base station to another. The connection also eventually changes between the two MEC hosts.

The specifications of 3GPP and ETSI describe how to secure the connections between the user and the MEC host and between MNO and the MEC host. These descriptions also include the mobility of MEC hosts. However, the specifications do not address the security and privacy issues when the user establishes a connection to the MEC application and changes from one MEC host to another. Therefore, we address the problem of establishing a secure communication path between the MEC application user in the UE and the MEC application in the MEC host, considering both when the user is static and when the user is mobile. This paper covers only the case where the MNO does not change during the connection.

Another problem we tackle is how to protect the privacy of the user against outsiders and also against the MEC host, MNO, and MEC applications. The outsiders can be considered Dolev–Yao adversaries, which can observe, replay, delete, fabricate, and reorder the messages in the communication channel [38,39]. This type of adversary cannot decrypt the encrypted messages unless it has the correct key [40]. The MNO, MEC host, and APPIFY can be considered honest-but-curious adversaries. Such adversaries are legitimate parties in the system and follow the protocol faithfully. Meanwhile, they may try to gather as much information as possible about the user from the traffic they see [41,42]. It is reasonable to assume that these parties are honest because they have a high risk of losing their reputation and business if they act maliciously and are caught.

## 4. AKMA and Its Privacy Enhancement

Authentication and key management for applications (AKMA) is described in 3GPP TS 33.535 [14]. In this section, we first present the AKMA protocol in detail and provide its formal verification using the ProVerif tool. Second, we present a new attack against AKMA, along with a countermeasure against the newly found attack. Third, we introduce privacy-enhanced AKMA (PE-AKMA), which is later used as a building block in the protocols of Section 5. In addition, we also provide formal verification of the PE-AKMA protocol in ProVerif. Finally, we show how the privacy-enhanced AKMA can be adapted to AKMA profiles for transport layer security (TLS)-based protocols, described in [14,43]. The source codes of the formal verifications of AKMA and PE-AKMA are publicly available in [44].

### *4.1. Authentication and Key Management for Applications*

Several functions in the core of the mobile network take part in AKMA. The first one is unified data management (UDM), which stores information about the subscribers of the MNO. Second, the AKMA anchor function (AAnF) manages temporary information about subscribers and generates temporary session keys. The third function is the authentication server function (AUSF), which provides the connection between UDM and AAnF. The AUSF also generates AKMA key material with the 5G authentication vector. Fourth, the network exposure function (NEF) establishes the connection between AAnF and the application function (AF) in case AF is not located inside the home network of the MNO [45]. In the rest of the paper, we use the term MNO to cover all functions in the core of the mobile network that participates in the AKMA service.

The Application Function (AF) depicts the online services or applications that the user is interested in. The AF is authenticated and authorized by MNO before providing the key that is shared with the UE. If AF is located inside the network of the MNO, then AF communicates directly with AAnF, while external AF requires NEF for this communication [14].

During AKMA, MNO shares the subscription permanent identifier (SUPI) with AF if it is inside the network of the MNO [14]. SUPI is a globally unique identifier of the subscriber, and it is only used inside the 3GPP system [46].

If AF is outside the network of the MNO, then MNO shares the generic public subscription identifier (GPSI) with AF [14]. GPSI addresses the subscriber in different data networks and is either the phone number, i.e., mobile subscriber international subscriber directory number (MSISDN), or some external identifier. The 3GPP system stores the association between the GPSI and the SUPI, but there is no implication of a one-to-one relationship between these two identifiers [46].

The AKMA service key, $K_{AKMA}$, is derived from the shared key that UE and the 5G mobile network obtain as a result of the network access (primary) authentication between them. In addition, the AKMA key identifier (A-KID) is assigned to the user to identify the key $K_{AKMA}$. A-KID is composed of the AKMA temporary UE identifier (A-TID) and the identifier of the mobile network operator, ID-MNO. The application-specific key $K_{AF}$ is derived from $K_{AKMA}$ and ID-AF by both UE and MNO, where ID-AF is the identifier of AF, and is essentially the fully qualified domain name (FQDN) of the AF [47]. The key $K_{AKMA}$ and its identifier A-KID are updated with every primary authentication, but the lifetime of $K_{AF}$ depends on the policy of the MNO. For example, after $K_{AKMA}$ is renewed, UE and AF can still use $K_{AF}$ to secure their communications until the lifetime of $K_{AF}$ expires or the application session ends. Afterward, a new $K_{AF}$ will be derived from the new $K_{AKMA}$ [14].

The AKMA also allows the anonymous user access to an AF residing inside the mobile network, which is used when AF does not require UE identification [14]. If the AF sends to the MNO a key request indicating that the user should be anonymous, then, in the reply, the AF receives only the shared key $K_{AF}$ and the expiration time of that key. Afterwards, the AF uses A-KID as a temporary user identifier of the UE.

Next, we present the steps of AKMA in Protocol 1, and Figure 3 summarizes the communication flow of the AKMA protocol, which is adapted from 3GPP TS 33.535 [14].

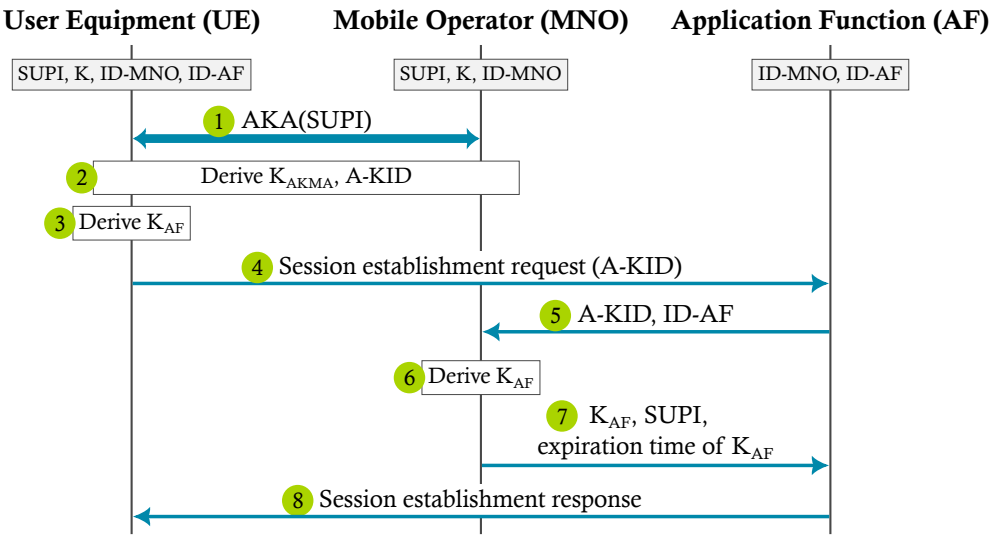


**Figure 3.** Authentication and key management for applications (AKMA), adapted from [14].

**Protocol 1** Authentication and key management for applications (AKMA)

Goal. Providing authentication and key sharing between UE and AF based on the subscription of UE with MNO.

*The protocol:*

**Authentication and key agreement (AKA):**

1. UE runs AKA with MNO. (If an existing and valid shared key $K_{AUSF}$ is stored, this step can be skipped.)
2. UE and MNO derive $K_{AKMA}$ from the shared key $K_{AUSF}$ and generate the AKMA Key Identifier (A-KID). Here, A-KID is composed of A-TID and ID-MNO.

**Derivation of credentials in UE:**

3. UE derives $K_{AF}$ for the AF from the $K_{AKMA}$ and ID-AF.

**Session Establishment Request:**

4. UE sends a session establishment request with the parameter A-KID to AF.

**Fetching UE credentials:**

5. AF sends a request to MNO for the application key by including A-KID and its own identifier ID-AF.
6. MNO retrieves the $K_{AKMA}$ related to the A-KID and derives $K_{AF}$ from the $K_{AKMA}$ and ID-AF.
7. MNO sends the SUPI or GPSI of UE, $K_{AF}$, and the expiration time of $K_{AF}$ to AF.

**Session Establishment Response:**

8. AF sends a session establishment response to UE after obtaining the key.

### *4.2. Formal Verification of AKMA*

In this section, we explain our ProVerif model of the AKMA protocol and give the results of its formal verification, as well as their analysis.

#### 4.2.1. Threat Model

We assume the presence of the Dolev–Yao adversary in the unsecured channel (between UE and AF). The adversary can observe, replay, delete, inject, and reorder the messages in the channel; it cannot decrypt encrypted messages or forge signatures without capturing appropriate keys [20,38,39]. The adversary cannot compromise the secure channels [20].

#### 4.2.2. Constructing the Protocol

Our ProVerif model of AKMA protocol is available online [44]. While constructing the protocol, we took the following into account.

- We assume that AKA is completed successfully between UE and MNO before AKMA starts. UE and MNO share SUPI and K, and as a result of AKA, they also share $K_{AKMA}$.
- In AKMA specifications, the protocol runs through the middle intermediate nodes (AUSF, AAnF, NEF) of MNO. In our implementation, these intermediate nodes are not explicitly defined. Instead, MNO is modeled as a single entity.
- We do not include in our model the details of the key derivation of $K_{AKMA}$ from K [14]. We simply define a function that derives $K_{AKMA}$ from K.
- The channel between UE and MNO is secure since they have completed AKA.
- The channel between MNO and AF is secure [48]. If AF resides inside the mobile network, the channel can be assumed to be secure. If AF resides outside of the mobile network, then MNO and AF must authenticate each other, e.g., based on the client and server certificates, and establish a TLS connection .
- The channel between UE and AF is not secure.

The queries in the ProVerif code are designed to check security and authentication properties which are explained in Section 2.4. We provide details of the queries and how we constructed the AKMA protocol model in ProVerif in Appendix E.

#### 4.2.3. Results

The result of the security properties we get when we run the protocol "AKMA.pv" [44] is summarized in *AKMA* column Table 1.

**Table 1.** Summary of security properties for AKMA and PE-AKMA.

| **Security Properties** | | **AKMA** | **PE-AKMA** |
|---|---|---|---|
| Secrecy | SUPI | True | True |
| | K | True | True |
| | A-KID | **False** | **True** |
| | $K_{AKMA}$ | True | True |
| | $K_{AF}$ | True | True |
| Strong Secrecy | SUPI | True | True |
| | K | True | True |
| Weak Secrecy | SUPI | True | True |
| | K | True | True |
| Forward Secrecy | in UE | **False** | **True** |
| | in MNO | False | False |

The secrecy result can be interpreted as (1) A-KID could be captured by the attacker, (2) the long-term key K, long-term identifier SUPI, AKMA key $K_{AKMA}$, and AF specific session key $K_{AF}$ are not learned by the attacker during the protocol.

The result of strong secrecy means that the attacker cannot distinguish whether the SUPI and K differ in different sessions. In addition, we can interpret the result of weak secrecy in such a way that AKMA does not enable brute-force attacks, e.g., dictionary attacks.

Keeping in mind that ID_AF is public, we can interpret the result for forward secrecy in the following way. If the attacker obtains K and SUPI either from the UE or MNO, then this attacker can derive the session keys $K_{AF}$ that are used to secure past communications. This is the assumption that the attacker recorded the past protocol messages exchanged over the public channel (including messages of the AKA protocol). Please note that the leakage of the secret key at the UE is shown to be possible via side-channel attacks [49].

Next, we discuss queries to check the authentication properties of AKMA. The results of running these queries are summarized in Table 2. As already mentioned in Section 2.2, AKMA, in itself, does not aim to provide authentication between UE and AF. However, for completeness, we have queries for checking authentication properties between those parties in the ProVerif model.

First, we check the aliveness of the parties. The result of the queries of aliveness is false for (1) UE with AF and true for (2) AF with UE, (3) MNO with AF, and (4) AF with MNO. If the attacker captures the messages of UE for AF and impersonates AF, UE completes the protocol without AF realizing it. Therefore, AKMA does not guarantee the aliveness of AF to UE. On the other hand, only the correct UE can send a valid A-KID, and if AF finds out in the dialogue with MNO that the A-KID is valid, AF can be sure that UE has at least started the protocol. Thus, AKMA guarantees the aliveness of UE to AF.

Second, we check queries for weak agreement. The result is that the MNO has a weak agreement with AF, and AF has a weak agreement with MNO, but AF does not have a weak agreement with UE. ProVerif has found the following attack. A-KID is sent in a public channel, and the Dolev–Yao adversary captures it. Afterward, the attacker inserts messages to the communication channel of UE and AF such that (1) AF receives a request even before UE sends the request or (2) UE receives a response from AF even before AF sends the response.

Third, we check non-injective agreement. The non-injective (and consequently injective) agreements do not hold between UE and AF because we have already seen that they do not obtain even weak agreements. Note that this is not a major shortcoming for

AKMA because it is assumed to be used together with some follow-up protocol between UE and AF.

**N**on-injective agreements hold for AF to MNO on $K_{AF}$ and A-KID, and MNO to AF on A-KID. However, the non-injective (and consequently injective) agreement for MNO to AF on $K_{AF}$ does not hold. This is because ProVerif found a counter-example based on the fact that the protocol ends before MNO can verify whether AF has received the key $K_{AF}$.

Finally, ProVerif replies with "cannot be proved" to the queries for the injective agreements between AF to MNO on both A-KID and $K_{AF}$ and from MNO to AF on A-KID. This result means that ProVerif was not able to find proof that the injective agreement holds but, on the other hand, was not able to find a counter-example that would show it does not hold.

Yang et al. [45] prove the injective (and, therefore, also non-injective) agreement of MNO with AF and AF with MNO for A-KID and $K_{AF}$ using Tamarin. This partial contradiction between our and their results is because of the following. They have modeled the channel between MNO and AF not just as a secure channel but also as a reliable channel. Hence, the MNO gets confirmation of AF receiving the last message, which includes $K_{AF}$. In our ProVerif model, there is no such confirmation.

**Table 2.** Summary of authentication properties for AKMA.

| **Authentication Properties** | **UE->AF** | **AF->UE** | **AF->MNO** | **MNO->AF** |
|---|---|---|---|---|
| Aliveness | False | True | True | True |
| Weak Agreement | False | False | True | True |
| Non-Injective Agreement (A-KID) | False | False | True | True |
| Non-Injective Agreement ($K_{AF}$) | False | False | True | False |
| Injective Agreement (A-KID) | False | False | – | – |
| Injective Agreement ($K_{AF}$) | False | False | – | – |

### *4.3. Analysis of AKMA and Its Formal Verification*

The purpose of AKMA is to enable authentication between the user equipment (UE) and the application function (AF) and help them share a secret key using the MNO subscription. Therefore, we focus our formal verification on the authentication properties between UE and AF and between AF and MNO. We do not prove any agreement between UE and MNO with the formal verification. This is because: (1) before the AKMA protocol begins, UE and MNO have already completed primary authentication (5G AKA). Hence they have mutual authentication outside of our modeling; (2) they do not exchange messages during AKMA protocol.

We found the following issues related to A-KID being captured by the attacker.

- Revealing A-KID in public causes linkability issues, as already pointed put in Yang et al. [45]. The same A-KID means the same $K_{AKMA}$, which leads to the same SUPI, i.e., the same user. Unless the primary authentication is repeated, A-KID remains the same and continues to be used by the same user. Tracking the A-KID makes it easy to link the same user using many other applications.
- The attacker can send the A-KID to several different AFs, which UE does not intend to use, and can cause DoS attacks. In addition, this implies (honest-but-)curious MNO would get the wrong view about the AFs that UE runs AKMA with, and would draw an incorrect profile for UE.
- We found a new attack which is discussed in Section 4.4.

Another privacy issue with AF is that it learns SUPI or GPSI. As explained in Section 2.2, SUPI is sent to AF if AF is in the MNO network, and GPSI is sent otherwise. However, GPSI is MSISDN or an external identifier assigned to UE. Therefore, the same GPSI is sent to all the AFs that UE wants to connect to. If AFs share information, they can observe that the same user is using those other applications.

### *4.4. New Attack and Its Countermeasure*

We will now describe a spoofing attack, where a malicious AF impersonates another AF towards a UE. As already explained in Section 2.2, we use the term MNO to cover all network functions that participate in the AKMA service.

3GPP TS 33.535 [14] Section 6.3 specifies how an external AF obtains the application key $K_{AF}$ from MNO. AF and MNO establish a TLS connection during this process, and AF sends the request for $K_{AF}$. The request message includes A-KID and ID-AF. The specification states that the MNO shall check whether it can provide the service to the AF based on the configured local policy or based on the authorization information available in the signaling. But it does not explicitly require the MNO to verify that the AF is authorized to use the ID-AF in the request. This opens the door for the following attack.

Assume that two application functions, AF1 and AF2, can both connect to MNO. Assume also that AF2 listens to the traffic between UE and AF1 and then captures the A-KID of the UE. Now, AF2 sends the A-KID and the identity of AF1 to MNO (Step 5 in Figure 3). If MNO does not verify that the received identity belongs to the correct AF, it will provide $K_{AF}$ to AF2. Now, AF2 can pretend to be AF1 towards the UE.

A different but related attack, where the attacker impersonates UE towards AF, is presented in [45]. This attack is due to the lack of weak agreement from AF to UE.

To prevent the attack where AF2 impersonates AF1 towards a UE, the MNO needs to verify the received ID-AF before deriving the $K_{AF}$ in Step 6 in Figure 3. We informed a 3GPP delegate about both the attack and its fix.

The predecessor of AKMA, generic bootstrapping architecture (GBA), does not face a similar problem. The 3GPP TS 33.220 [11] states that "with the key material request, the NAF shall supply a NAF-ID (which includes the NAF's FQDN that the UE has used to access this NAF and the Ua security protocol identifier) to the BSF. The BSF shall verify that the NAF is authorized to use that FQDN".

### *4.5. Description of PE-AKMA*

Based on the formal verification results of AKMA in Section 4.2 and the subsequent discussions in Section 4.3, we developed a privacy-enhanced AKMA. The PE-AKMA protocol employs key and data encapsulation mechanisms (KEM and DEM, respectively) to protect the data that UE sends via the Application Function (AF) to MNO. The KEM and DEM concepts are explained in Appendix C.

Recall that the MNO, the UE, and the AF participate in the AKMA procedure. The MNO has identifiers of its subscribers (SUPI) and long-term keys (K). The MNO also has identifiers (ID-AF) of different AFs. In the privacy-enhanced AKMA (PE-AKMA), the MNO also has public and secret key pairs ($PK_{MNO}$, $SK_{MNO}$), and $K_{MNO}$.

The owner of the UE, Alice, is a subscriber of MNO, and she has an identifier SUPI for MNO, long-term key, K, and the public key of MNO, $PK_{MNO}$. Alice also has a user identifier, user_id, and password, psw, for AF. During the PE-AKMA, Alice uses another key, locally generated $K_{UE}$, to compute a pseudonym to be used instead of her AF user identifier in the protocol.

The application function may reside in the mobile network or connect to the mobile network from outside. Either way, MNO authenticates and authorizes AF before providing the AKMA session key. The AF has an identifier ID-AF, and it knows the identifier ID-MNO of the MNO.

Figure 4 presents the privacy-enhanced AKMA protocol. The figure is adapted from 3GPP TS 33.535 [14]. The steps with red text are our additions to improve the privacy of the AKMA procedure. The PE-AKMA is explained in Protocol 2.

**Protocol 2** Privacy-enhanced authentication and key management for applications (PE-AKMA)

Goal. Improve the privacy of AKMA considering the shortcomings discussed in Section 4.3.

*The protocol:*

**Authentication and Key Agreement (AKA):**

1. UE runs AKA with MNO. (If an existing and valid shared key $K_{AUSF}$ is stored, this step can be skipped.)
2. UE and MNO derive $K_{AKMA}$ from the shared key $K_{AUSF}$ and generate the AKMA key identifier (A-KID). Here, A-KID is composed of A-TID and ID-MNO.

**Derivation of credentials in UE:**

3. (a) UE derives $K_{AF}$ for the AF from the $K_{AKMA}$ and ID-AF.
   (b) UE computes T=$HMAC_{K_{UE}}$(user_id, ID-AF) to conceal the user_id.
   (c) UE prepares the message M = A-KID ∣ ID-AF ∣ T.
   (d) UE starts encapsulation as described in Appendix C. UE applies Key-Encaps($PK_{MNO}$) to obtain a ciphertext C and a shared key $K_{sh}$.
   (e) UE saves the key $K_{sh}$.
   (f) UE applies Data-Encaps($K_{sh}$, M) to obtain N.
   (g) UE derives $K'_{AF}$ = KDF($K_{sh}$, $K_{AF}$), where KDF is the key derivation function. This key will be the shared AKMA key between AF instead of $K_{AF}$.

**Session Establishment Request:**

4. UE sends a session establishment request to AF with the parameters N, C, and ID-MNO.

**Fetching UE credentials:**

5. AF sends a request to MNO for the application key by including N, C, and its own identifier ID-AF.
6. (a) MNO retrieves the $K_{AKMA}$ related to the A-KID and derives $K_{AF}$ from the $K_{AKMA}$ and ID-AF.
   (b) MNO starts decapsulation as described in Appendix C. MNO applies Key-Decaps($SK_{MNO}$, C) and obtains $K_{sh}$ as an output.
   (c) MNO computes M=Data-Decaps($K_{sh}$, N), where M = A-KID ∣ ID-AF ∣ T.
   (d) MNO verifies that three ID-AFs (one that MNO has, one that UE included in M, and one that AF sent) are the same to ensure that the request of UE is sent from the correct AF.
   (e) MNO acquires the $K_{AKMA}$ related to A-KID and derives $K_{AF}$ using ID-AF.
   (f) MNO derives $K'_{AF}$ = KDF($K_{sh}$, $K_{AF}$).
   (g) MNO computes a hashed identifier H-ID=$HMAC_{K_{MNO}}$(T ∣ SUPI ∣ ID-AF) to replace SUPI and GPSI.
7. MNO sends H-ID, $K'_{AF}$, and the expiration time of $K'_{AF}$ to AF.

**Session Establishment Response:**

8. AF sends a session establishment response to UE after obtaining the key.

### *4.6. Formal Verification of PE-AKMA*

This section presents the formal verification of PE-AKMA using ProVerif.

#### 4.6.1. Constructing the Protocol

The construction of the PE-AKMA protocol in ProVerif is similar to what we explained in Section 4.2, except for the new variables and the implementation of the key and data encapsulation mechanisms (KEM/DEM) explained in Appendix C in the PE-AKMA.

The queries in the ProVerif code are designed to check security and authentication properties, explained in Section 2.4. We provide details of the queries and how we constructed the PE-AKMA protocol model in ProVerif in Appendix F.

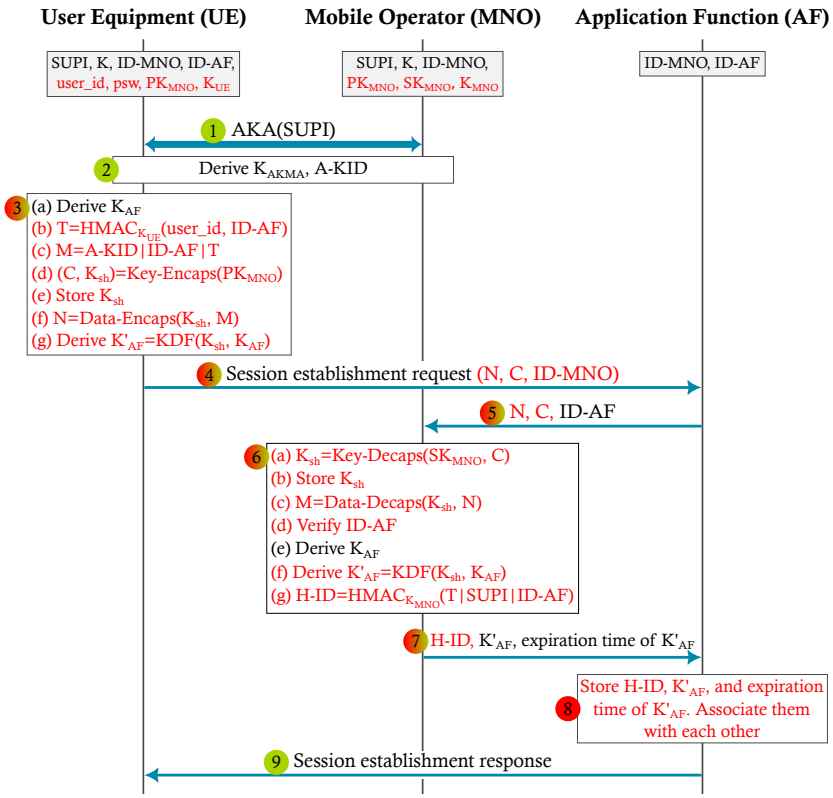


**Figure 4.** Privacy-enhanced AKMA protocol.

#### 4.6.2. Results

Table 1 summarizes the security properties in AKMA and in PE-AKMA. The highlighted rows show that PE-AKMA prevents attackers from capturing A-KID and provides forward secrecy in UE.

Now, we explain the security properties of the new variables in PE-AKMA. In addition to the secrecy queries that were verified in the generic AKMA protocol, we have shown that the attacker cannot learn $SK_{MNO}$, $K_{sh}$, and $K'_{AF}$.

The PE-AKMA protocol provides forward secrecy in the case where the secrets, K and SUPI, are leaked. That is because the key $K_{sh}$ remains secret. Consequently, past $K'_{AF}$ remains safe from the attacker. Please note that if SK_MNO leaks, then the forward secrecy is broken. However, we argue that a compromise of the parameters in the UE, i.e., K and SUPI, is more likely than a compromise of the long-term secret keys in the MNO.

We defined queries to check the authentication properties of PE-AKMA. The results are summarized in Table 3.

**Table 3.** Summary of authentication properties for PE-AKMA.

| Authentication Properties | UE->AF | AF->UE | AF->MNO | MNO->AF |
|---|---|---|---|---|
| Aliveness | False | True | True | True |
| Weak Agreement | False | False | True | True |
| Non-Injective Agreement (N) | False | False | True | True |
| Non-Injective Agreement ($K'_{AF}$) | False | False | True | False |
| Injective Agreement (N) | False | False | – | – |
| Injective Agreement ($K'_{AF}$) | False | False | – | – |

Aliveness holds for AF, with UE as the responder, while the weak agreement does not. Neither aliveness nor weak agreement holds for UE with AF as the responder. Please note that aliveness in this context means that AF is assured of the following. AF has a dialogue

with MNO about certain UE and that UE has at least started the protocol. In contrast to AKMA, the AF does not obtain the SUPI (or GPSI) that identifies the UE in the mobile network. Instead, the AF obtains H-ID, i.e., a pseudonym of the UE.

ProVerif proves that MNO has a weak agreement with AF and AF has a weak agreement with MNO.

We introduced new keys and identifiers. Therefore, we have new queries for non-injective and injective agreements.

ProVerif shows that AF has a non-injective agreement with MNO on $K'_{AF}$ and on N. In addition, MNO has a non-injective agreement with AF on N. However, MNO does not have a non-injective agreement with AF on $K'_{AF}$ because the protocol ends before MNO can verify that AF has the key.

On the other hand, ProVerif cannot prove the injective agreements for AF and MNO, with similar reasons explained in Section 4.2.

Finally, the lack of weak agreement between the UE and AF implies that non-injective and injective agreements between UE and AF do not hold as well. We remark that the weak and non-injective agreements between the UE and AF would hold if a key confirmation round is added at the end of the protocol. Similar to AKMA, this is a minor shortcoming of PE-AKMA because it is assumed to be used with some follow-up protocol between UE and AF.

### *4.7. Remarks on the PE-AKMA*

We improved AKMA considering the issues that are discussed in Section 4.3.

We added Step 6(d) in Protocol 2, where ID-AF is verified by the MNO, as a countermeasure against the attack described in Section 4.4. Even if a malicious AF captures N and C and sends those to MNO together with the ID-AF of some other AF, then, because of this countermeasure, the request will not be accepted by MNO.

We introduced KEM/DEM, explained in Appendix C, in the privacy-enhanced AKMA. In our threat model, we assume the presence of an active attacker in the channel between the UE and the AF. Thus, we recommend using KEMs providing resistance against such attackers, i.e., at least has the indistinguishability under chosen-plaintext attack (IND-CPA) property [50].

Usually, IND-CPA KEMs are built using (randomized) probabilistic encryption schemes [51]. Hence, we implemented the randomized KEMs in our ProVerif model. For the DEM, we use symmetric encryption with the key resulting from the KEM. To ensure further security properties [52], we equip the DEM with a message authentication code (MAC) check (Encrypt-then-MAC), which is the case in the standardized elliptic curve integrated encryption scheme (ECIES) [48].

We used the KEM/DEM paradigm to protect the message M in Step 3 in Figure 4 and to provide forward secrecy. Please note that using TLS with the AKMA protocol would provide forward secrecy, but with the introduction of KEM/DEM, we guarantee forward secrecy at the UE, also in those cases when UE and AF do not use TLS. In PE-AKMA, we use $K'_{AF}$ instead of $K_{AF}$ as a session key. $K'_{AF}$ is derived using freshly generated public and secret key pairs in UE. Therefore, even if K and SUPI are leaked, attackers cannot decrypt the earlier communication.

Our enhanced protocol can be used when it is not desirable that the MNO is able to learn the user_id in AF. To achieve this, we introduced a new identifier, T, which is derived by computing the keyed hash of user_id and ID-AF. The T is given to the MNO instead of the user_id as a unique identifier for the user. Then, the MNO does not learn the UE identifier related to this specific AF. Moreover, even if the user has the same user_id for different AFs, the MNO would not be able to recognize this since T also depends on the AF identifier.

If the application function does not require a user_id, then UE can compute T with a null value instead of the user_id in Step 3(b) of Figure 4.

We also recall that AKMA specifications (Section 6.6 of TS 33.535 [14]) include a case for anonymous user access. In order to use it, AF should indicate the anonymous user access request while sending the key request to MNO, Step 5 of Figure 3. Anonymous user access can also be adapted to PE-AKMA, such that AF indicates the request in Step 5 of Figure 4. Then, in Step 7, MNO does not send H-ID to AF. In this case, N is used as a temporary user identifier.

In addition, the user might want to stay anonymous towards the AF. To achieve this anonymity, the user prepares M in Step 3(c) of Figure 4 such that T is a null value. When MNO decrypts the message in Step 6(c) of Figure 4, it understands the anonymity request of the user by checking T and does not send H-ID later in Step 7. Similarly to the earlier case, N is used as a temporary user identifier. Note that N is different every time UE sends the session establishment request; therefore, AF does not learn that the two sessions belong to the same user.

Another feature of our protocol is based on the fact that in the standard AKMA, the MNO provides the UE identifiers, i.e., SUPI or GPSI, to AF along with the shared key $K_{AF}$. Such identifiers can be used to track the user. Therefore, we replaced SUPI/GPSI with H-ID to protect these identifiers from AF. H-ID is derived by computing the keyed hash of SUPI, T, and ID-AF. Therefore, H-ID is a unique identifier depending on UE, MNO, and AF. Note that AF does not even learn whether two user_ids belong to the same user.

### *4.8. AKMA Profiles for TLS*

The specification of AKMA includes two profiles for applying AKMA when the UE connects to AF over TLS: (1) shared key-based UE authentication with certificate-based AF authentication; and (2) shared key-based mutual authentication between UE and AF [14]. They are based on the TLS profiles in the GBA specification [43].

This section shows how to apply the privacy-enhanced AKMA from Section 4.5 in profiles (1) and (2). The numbering of the steps of the profiles conforms with the numbering in the specifications [14,43]. The changes to the original profiles are indicated by red color in Figures 5 and 6.

#### 4.8.1. Profile 1: Shared Key-Based UE Authentication with Certificate-Based AF Authentication

This profile explains how AF is authenticated by UE with its certificate, while UE is authenticated later with the shared key resulting from AKMA. The procedure is described below in Protocol 3 and shown in Figure 5.

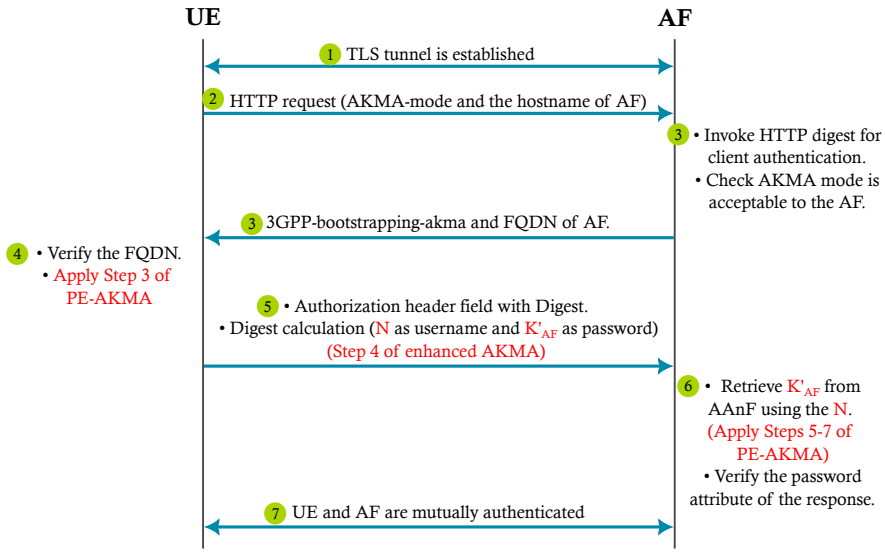


**Figure 5.** Profile 1 with PE-AKMA.

**Protocol 3** AKMA Profile 1 for TLS with PE-AKMA

Goal. Adapting PE-AKMA to the AKMA Profile 1 for TLS-based protocols.

*The protocol:*

**AF authentication for UE:**

1. - TLS tunnel is established between UE and AF.
   - The user authenticates the AF with its public key certificate, while UE is not authenticated by AF yet.

**UE authentication for AF:**

2. - UE sends an HTTP request to AF inside the TLS tunnel to establish the HTTPS connection.
   - UE indicates that AKMA-based authentication is supported in the request.
   - UE also includes the hostname of the AF in the HTTP header.
3. - AF invokes HTTP digest with UE to perform client authentication.
   - AF also checks if the product tokens in Step 2 indicate AKMA mode acceptable to AF.
   - AF sends the constant string 3gpp-bootstrapping-AKMA and its fully qualified domain name (FQDN) to UE.
4. - UE verifies the FQDN of AF.
   - UE applies Step 3 of the PE-AKMA (Protocol 2).
5. - UE sends the authorization header field with Digest to AF.
   - The digest calculation includes N as the username (instead of A-KID) and $K'_{AF}$ as the password (instead of $K_{AF}$).
   - This step corresponds to Step 4 of the PE-AKMA (Protocol 2).
6. - AF retrieves $K'_{AF}$ from AAnF (MNO) using N.
   - Steps 5–7 of PE-AKMA are run for the key retrieval.
   - AF verifies the password attribute of the response after receiving the $K'_{AF}$ from AAnF.

**Secure communication:**

7. UE and AF communicate over a mutually authenticated secure TLS tunnel.

#### 4.8.2. Profile 2: Shared Key-Based Mutual Authentication between UE and AF

The second profile uses the shared key resulting from AKMA for mutual authentication. This profile does not require any certificate from the parties because it is based on Transport Layer Security Pre-Shared Key (TLS-PSK), about which more information is provided in Appendix D. The protocol is described in Protocol 4 and shown in Figure 6.

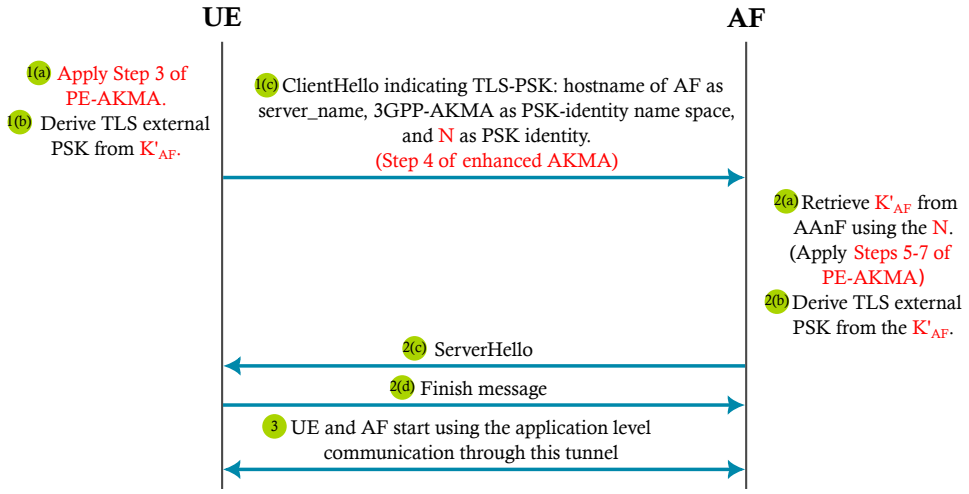


**Figure 6.** Profile 2 with PE-AKMA.

**Protocol 4** AKMA Profile 2 for TLS with PE-AKMA

Goal. Adapting PE-AKMA to the AKMA Profile 2 for TLS-based protocols.

*The protocol:*

**TLS-PSK request:**

1. (a) UE starts with preparing AKMA tokens by applying Step 3 of the PE-AKMA (Protocol 2).
   (b) UE also derives the TLS external PSK from $K'_{AF}$.
   (c) UE sends a ClientHello message indicating the TLS-PSK.
   - The ClientHello message includes the hostname of AF as the server_name, 3GPP-AKMA as the PSK-identity namespace, and N (instead of A-KID) as the PSK-identity.
   - This step corresponds to Step 4 of the PE-AKMA.

**TLS-PSK response:**

2. (a) AF retrieves $K'_{AF}$ from AAnF (MNO) using N. Steps 5–7 of PE-AKMA are run for the key retrieval.
   (b) AF derives the TLS external PSK from the $K'_{AF}$ after receiving $K'_{AF}$ from AAnF.
   (c) AF sends the ServerHello message, including the AKMA PSK identity.
   (d) UE replies with the Finish Message.

**Secure communication:**

3. UE and AF start using the TLS tunnel for application-level communication.

### *4.9. Feasibility of PE-AKMA and Profiles*

Comparing AKMA with PE-AKMA, we first recall that the message flow and the number of exchanged messages between the parties are similar for both protocols. For the communication overhead, the request messages (Steps 4 and 5) in PE-AKMA consist of N, C, and ID-MNO instead of A-KID in AKMA. Please note that C is the ciphertext of KEM, while N is the message resulting from symmetric encryption of M=A-KID | ID-AF | T, where $T=HMAC_{K_{UE}}$(user_id, ID-AF).

To illustrate the size of these parameters, we should mention that 3GPP standardized the use of the ECIES KEM/DEM for the SUPI encryption in 5G AKA, see 3GPP TS 33.501 [48] Annex C. The KEM ciphertext in ECIES is 32 bytes. Moreover, the DEM utilizes a 128 bits AES in counter mode. The HMAC algorithm, the mentioned 3GPP standard, uses the HMAC SHA 256. Consequently, in Steps 4 and 5, PE-AKMA creates approximately 1 KB of overhead compared to AKA. In addition, in Step 7, the change is sending H-ID instead of SUPI, which is sending 256 bits instead of 64 bits. In summary, the communication overhead for each message is less than 1 KB.

Considering the computation overhead, we compare AKMA and PE-AKMA with respect to the used cryptographic operations, i.e., hash functions (including key derivation functions, message authentication codes), public key operations (including encryption, decryption, KEM, signing), and symmetric key operations (including encryption, decryption, DEM). Computationally, public key operations are heavier than hash functions or symmetric key operations.

The 5G UE and MNO already have the capacity to execute KEM/DEM for concealing SUPI during AKA; see [48]. In addition, KEM/DEM mechanism is used only once per PE-AKMA. Please note that PE-AKMA is compatible with any KEM, e.g., the recently standardized post-quantum KEMs by the National Institute of Standards and Technology in the USA (NIST).

Considering the computational cost per entity, UE computes only three hash functions in AKMA, whereas in PE-AKMA, it computes two additional hash functions, one public key operation, and one symmetric key operation. AF does not have any computation in AKMA or PE-AKMA. The MNO computes three hash functions in AKMA, whereas PE-AKMA also computes two hash functions, one public key operation, and one symmetric key operation.

In summary, there are a few more symmetric and public key operations in PE-AKMA than in AKMA. However, these operations are standardized and are already used in 5G UE and 5G networks. Therefore, it is feasible to adapt PE-AKMA.

The communication and computational overheads in privacy-enhanced AKMA profiles, described in Section 4.8, compared to generic AKMA profiles, described in [14], are the same as the overheads of PE-AKMA over AKMA.

## 5. Protocols for Accessing to MEC Application

This section presents a solution to maintain security and privacy for the MEC applications intended for static and mobile users. The solution is based on adapting the privacy-enhanced AKMA and AKMA profiles, shown in Sections 4.5 and 4.8, respectively.

The setting can be illustrated by the following scenario. Alice is an MNO subscriber and uses an application called APPIFY. The application APPIFY appears both in the form of the main server of APPIFY (S-APPIFY) and as the local APPIFY server instance in the MEC host (M-APPIFY). Alice uses AKMA to authenticate herself to APPIFY.

The S-APPIFY has a key pair consisting of a public encryption key and a secret decryption key (EK, DK). The users of APPIFY utilize the public key EK to encrypt their identifiers. ID-APPIFY and ID-MNO are the identifiers of S-APPIFY and MNO, respectively, and are known by the users and also by S-APPIFY and MNO themselves. Alice has her subscription permanent identifier (SUPI) for the MNO context and identifier–password pair (Alice13, psw) for the APPIFY context.

Our solution consists of four protocols: 5A, 5B, 5C, and 5D. Protocols 5A and 5B are executed between Alice and the S-APPIFY. Protocol 5A is used to register a new user or a new trusted device to the main server of APPIFY. Protocol 5B is run for renewing the public key EK of S-APPIFY and shared keys related to the users. This protocol is also run to establish a secure connection between Alice directly and S-APPIFY for the purpose of providing services to Alice from the main server. Protocol 5C is executed between Alice and M-APPIFY when Alice wants to use services provided by APPIFY in the MEC host. Protocol 5D is used when Alice is moving while using APPIFY in the MEC host, and the serving MEC host and the M-APPIFY server have to be changed. The section ends with an analysis of our solution.

In Protocols 5A and 5B, a transport layer security (TLS) connection is established between Alice and the main server of APPIFY. One good solution is to use enhanced Profile 1, described earlier in Section 4.8.1.

A secure channel between Alice and MEC host [53], e.g., a TLS or datagram transport layer security (DTLS) connection, is established in Protocols 5C and 5D. This secure channel establishment is based either on a certificate of the MEC host or on a pre-shared key PSK method with a key that results from the run of AKMA between Alice and MEC, as with Profile 2, described in Section 4.8.2. We leave out of the scope of this paper further details of the secure channel between Alice and MEC.

Next, we explain the four protocols of our solution.

### *5.1. Protocol 5A: Signing Up*

Protocol 5A is run when a new user wants to sign up, or a registered user wants to add a new trusted device. In this context, a new device implies that the user also has a new SUPI. Protocol 5A runs between Alice and S-APPIFY through MNO. We use Profile 1 (Protocol 3 in Section 4.8.1) with PE-AKMA for key distribution between Alice and S-APPIFY. The protocol is explained below, and the communication flow is shown in Figure 7.

**Protocol 5 A:** Signing Up

Goal. Key sharing when signing up a new user or adding a new device by a registered user.

*The protocol:*

1. Alice runs AKA with MNO. (If an existing and valid shared key $K_{AUSF}$ is stored, this step can be skipped.)

**Establishing secure connection and authentication of S-APPIFY:**

2. Alice and S-APPIFY run Steps 1–3 of Profile 1 (Protocol 3 in Section 4.8.1). This step includes the TLS connection establishment and the HTTP request of Alice. Moreover, S-APPIFY starts client authentication via AKMA.

**Authentication of UE:**

3. Alice runs Step 4 of Profile 1, which includes verifying the FQDN of S-APPIFY and computing AKMA parameters, e.g., N and $K'_{AF}$. Note that a key derived from $K'_{AF}$ is used as a shared key instead of $K_{AF}$.
4. Alice runs Step 5 of Profile 1, i.e., sends a session establishment request including the parameter N as username and the key $K'_{AF}$ as password.
5. Alice then sends Alice13, psw, and a selected option to S-APPIFY through the secure channel. The options are (1) new user registration and (2) registering a new device.
6. S-APPIFY checks the database for Alice13 and psw.
   If Alice13 is not found in the database:
   If Option (1) is selected, then S-APPIFY stores Alice13 and psw in the database, and the protocol continues with Step 7.
   If Option (2) is selected, then the protocol continues with Step 7* with the message "either wrong username or wrong psw".
   If Alice13 is found in the database:
   If Option (1) is selected, then the protocol continues with Step 7* with the message "user identity is already chosen, please choose another user identity".
   If Option (2) is selected, we still have two cases:
   If psw is correct: The protocol continues with Step 7.
   If psw is not correct: The protocol continues with Step 7* with the message "either wrong username or wrong password".

7*. S-APPIFY sends the error message, and the protocol is aborted.

**Key Confirmation of $K_{AF}$ for S-APPIFY:**

7. S-APPIFY runs Step 6 of Profile 1 to obtain $K'_{AF}$ from MNO. Then, it verifies that this key matches with the one that UE sent earlier.
   UE and S-APPIFY are now able to communicate over mutually authenticated TLS channel (Step 7 of Profile 1).
8. (a) S-APPIFY stores the pair (H-ID, $K'_{AF}$), sent in Step 7, and associates it with Alice13 and psw.
   (b) S-APPIFY computes the hash $H(K'_{AF})$ as an identifier of the (H-ID, $K_{AF}$) pair to be used later in Protocol 5C in Section 5.3.
   (c) S-APPIFY computes $MAC_{K'_{AF}}(EK)$. Note that S-APPIFY already verified that Alice has the shared key, but Alice did not verify the converse.

**Key Confirmation of $K_{AF}$ for Alice:**

9. S-APPIFY sends Alice13, EK, and $MAC_{K'_{AF}}(EK)$ along with the success message to Alice over the TLS channel.
10. Alice verifies $MAC_{K'_{AF}}(EK)$.
    If the verification is successful: The protocol continues with Step 11.
    If the verification is a failure: The protocol continues with Step 11*.
11. Alice sends a success message to S-APPIFY. Further communication continues over TLS; for example, the profile settings can be updated.

11*. Alice sends an error message "MAC failure" to S-APPIFY and the protocol is aborted.

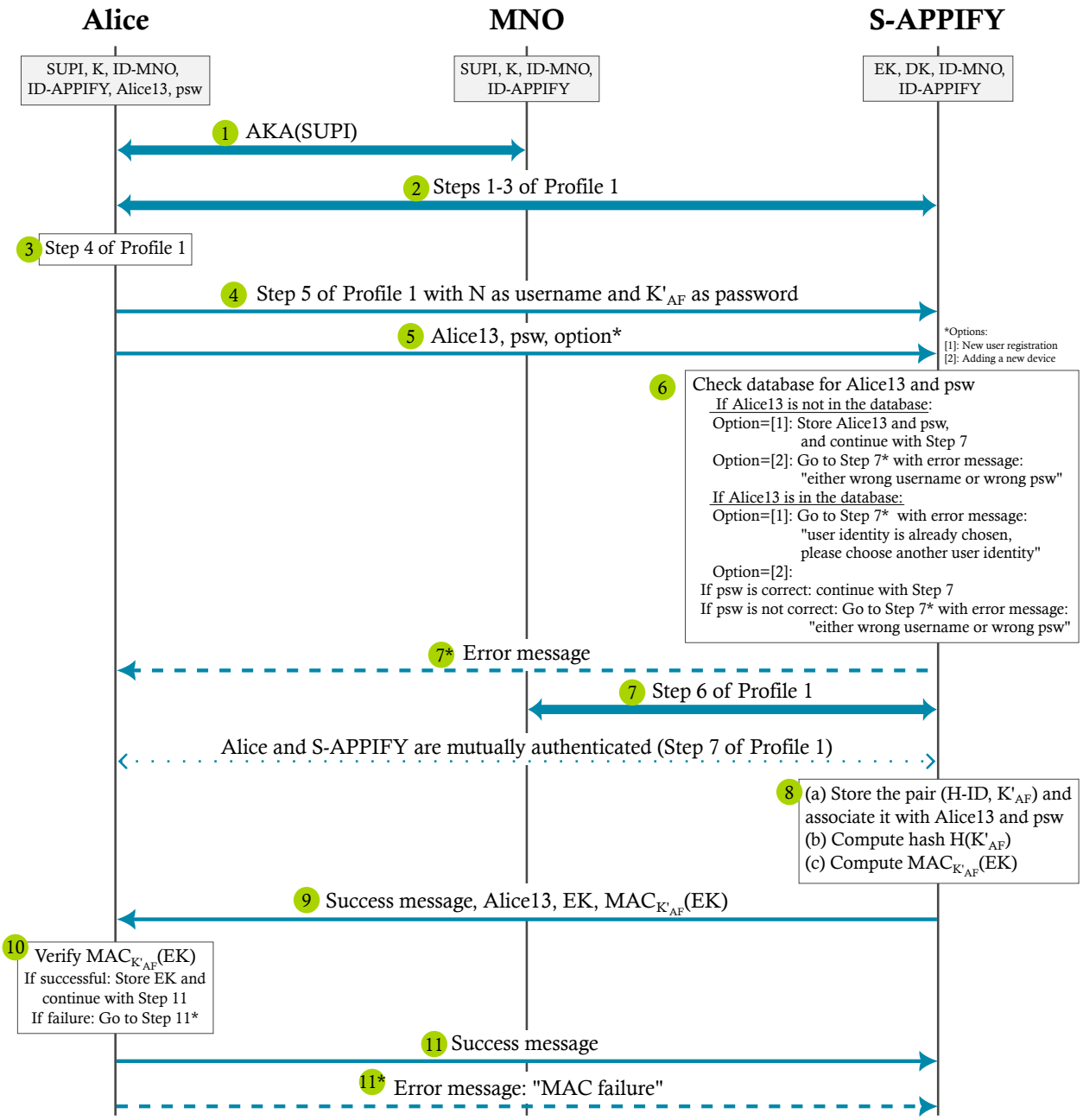


**Figure 7.** Communication flow of Protocol 5A: Signing Up.

### *5.2. Protocol 5B: Signing In*

Protocol 5B is run when the user and their devices are already registered and the user wants to connect to the main server. The user can renew the shared keys or use APPIFY from the main server. Protocol 5B runs between Alice and S-APPIFY and includes MNO. We use Profile 1 (Protocol 3 in Section 4.8.1) with PE-AKMA for key distribution between Alice and S-APPIFY. The protocol is explained below, and the communication flow is shown in Figure 8.

**Protocol 5 B:** Signing In

Goal. Key sharing, when the user and their devices are already registered and using the main server.

*The protocol:*

1. Alice runs AKA with MNO. (If an existing and valid shared key $K_{AUSF}$ is stored, this step can be skipped.)

**Establishing a secure connection and authentication of S-APPIFY:**

2. Alice and S-APPIFY run Steps 1–3 of Profile 1 (Protocol 3 in Section 4.8.1). This step includes the TLS connection establishment and the HTTP request of Alice. Moreover, S-APPIFY starts client authentication via AKMA.

**Authentication of UE:**

3. Alice runs Step 4 of Profile 1, which includes verifying the FQDN of S-APPIFY and computing AKMA parameters, e.g., N and $K'_{AF}$. Note that a key derived from $K'_{AF}$ is used as a shared key instead of $K_{AF}$.
4. Alice runs Step 5 of Profile 1, i.e., sends a session establishment request including the parameter N as username and the key $K'_{AF}$ as password.
5. Alice sends Alice13 and the selected option to S-APPIFY through the secure channel. The options are (1) Renewing EK and (2) Signing into the main server. In both options, $K_{AF}$ is updated.
6. (a) S-APPIFY checks the database for Alice13.
   If Alice13 is found in the database: The protocol continues with Step 6b.
   If Alice13 is not found in the database: The protocol goes to Step 9*.
   (b) S-APPIFY fetches the H-ID(s) related to the identity Alice13. Note that Alice may use the application from several different devices, and there would be different H-IDs for each device.

**Key Confirmation of $K_{AF}$ for S-APPIFY:**

7. S-APPIFY run Step 6 of Profile 1 to obtain $K'_{AF}$ from MNO. Then, it verifies that this key agrees with the one that UE sent earlier.
   UE and S-APPIFY are now able to communicate over mutually authenticated TLS channel (Step 7 of Profile 1).
8. (a) S-APPIFY checks if the H-ID sent by MNO matches any existing H-ID(s) found in Step 6b.
   If H-ID matches: The protocol continues with Step 8b.
   If there is an existing $K'_{AF}$, which is replaced by the new one.
   If there is no existing $K'_{AF}$, it is saved.
   If H-ID does not match: The protocol goes to Step 9*.
   (b) S-APPIFY computes the hash $H(K'_{AF})$ as an identifier of the (H-ID, $K'_{AF}$) pair to be used later in Protocol 5C.
   (c) S-APPIFY computes $MAC_{K'_{AF}}(EK)$.

**Key Confirmation of $K_{AF}$ for Alice:**

9. S-APPIFY sends EK and $MAC_{K'_{AF}}(EK)$ along with the success message to Alice.

9*. Error message "Either wrong username or authentication has failed" is sent by S-APPIFY, and the protocol is aborted.

10. Alice verifies $MAC_{K'_{AF}}(EK)$.
    If the verification is successful: UE stores EK and the protocol continues with Step 11.
    If the verification is a failure: The protocol goes to Step 11*.
11. Alice sends a success message to S-APPIFY. Further communication continues over TLS; for example, the profile settings can be updated.

11*. Alice sends an error message "MAC failure" to the main server of APPIFY, and the protocol is aborted.

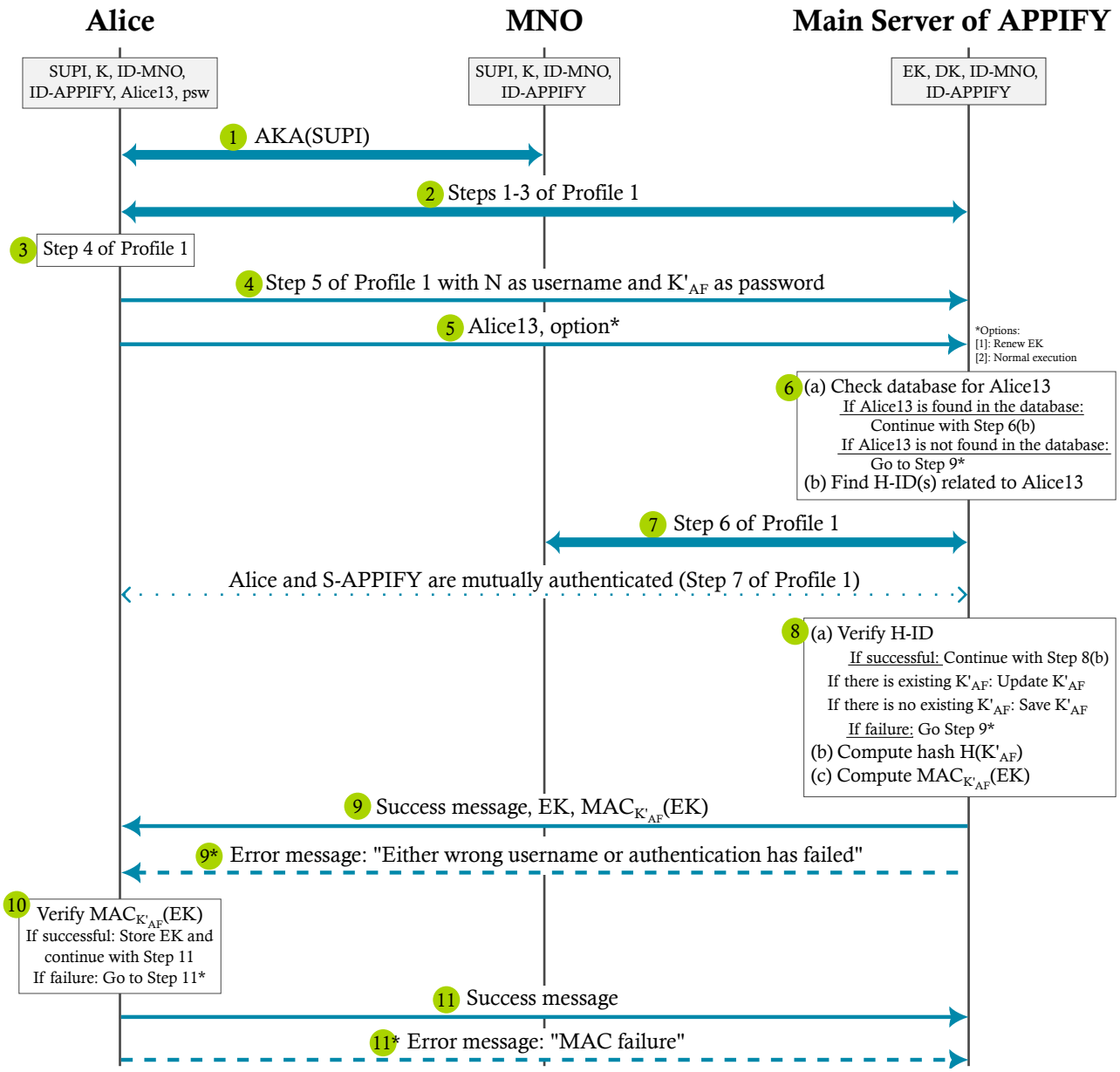


**Figure 8.** Communication flow of Protocol 5B: Signing In.

### *5.3. Protocol 5C: Connecting to Application in MEC Host*

Protocol 5C is run when at least Protocol 5A or Protocol 5B is run between Alice and S-APPIFY. Thus, Alice has up-to-date keys for the main server and wants to use M-APPIFY. The entities, MNO, MEC, M-APPIFY, and S-APPIFY, participate in this protocol. Protocol 5C is explained below, and the communication flow is shown in Figure 9.

At the beginning of the protocol, Alice has her identities SUPI for MNO and Alice13 for APPIFY. She also has psw, public key EK of S-APPIFY, and $K'_{AF}$. MNO knows the SUPI of Alice and has saved $K_{AKMA}$ and A-KID relating to the SUPI. S-APPIFY has the public key EK and saved Alice13, psw, (H-ID, $K'_{AF}$), and $H(K'_{AF})$ in the database. MEC and M-APPIFY do not have any information on the user.

**Protocol 5 C:** Connecting to MEC APPIFY

Goal. Establishing a secure and privacy-preserving connection with MEC APPIFY.

*The protocol:*

**Secure communication with MEC:**

1. Alice and MEC establish a secure connection, e.g., DTLS connection.

**Derivation of credentials in UE of Alice:**

2. (a) Alice computes the hash $H(K'_{AF})$.
   (b) Alice generates a random X.
   (c) Alice encrypts her identifier Alice13, X, and $H(K'_{AF})$ by using the public key of S-APPIFY EK: $M=E_{EK}$(Alice13, X, $H(K'_{AF})$).

**Request for the session with M-APPIFY:**

3. Alice sends the ID-APPIFY and M to MEC through the secure channel.
4. MEC notices the identifier of S-APPIFY and forwards the message M to M-APPIFY.
5. M-APPIFY forwards M to S-APPIFY.

**Distribution of session credentials:**

6. (a) S-APPIFY decrypts M with its secret key DK.
   (b) S-APPIFY then checks the database for Alice13.
   If Alice13 is in the database: S-APPIFY obtains the related identifiers and keys, e.g., $K'_{AF}$ that matches $H(K'_{AF})$.
   If there is an existing $K'_{AF}$, the protocol continues with Step 6c.
   If there is no existing $K'_{AF}$, the protocol continues with Step 6* with the error message "Run Protocol 5B".
   If Alice13 is not in the database: The protocol continues with Step 6* with the error message "User is not found".
   (c) S-APPIFY derives a new key $K_{AFD}=KDF(K'_{AF}, ID\text{-}M\text{-}APPIFY, Alice13, X)$, where the ID-M-APPIFY is the identifier of M-APPIFY.

6* The error message is sent, and the protocol is aborted.

7. S-APPIFY sends $K_{AFD}$, Alice13, X, the user profile of Alice13, and the expiration time of $K_{AFD}$ to M-APPIFY.
8. (a) M-APPIFY stores Alice13, $K_{AFD}$, and the expiration time of $K_{AFD}$.
   (b) M-APPIFY encrypts $ET=E_{K_{AFD}}$(expiration time of $K_{AFD}$).
   (c) M-APPIFY computes $Y=MAC_{K_{AFD}}(ID\text{-}M\text{-}APPIFY, Alice13, X)$.
9. M-APPIFY sends ID-M-APPIFY, Y, and ET to MEC.
10. MEC forwards ID-M-APPIFY, Y, and ET to Alice.
11. (a) Alice derives $K_{AFD}=KDF(K'_{AF}, ID\text{-}M\text{-}APPIFY, Alice13, X)$ after receiving ID-M-APPIFY.
    (b) Alice then decrypts $D_{K_{AFD}}$(ET) and stores the expiration time of $K_{AFD}$.
    (c) Alice computes $MAC_{K_{AFD}}(ID\text{-}M\text{-}APPIFY, Alice13, X)$ and verifies whether it matches Y.
    If the verification is successful: The protocol continues with Step 12.
    If the verification is a failure: The protocol continues with Step 12*.

**Secure communication with M-APPIFY:**

12. Alice and M-APPIFY establish a TLS connection with a pre-shared key (TLS-PSK), where the key is $K_{AFD}$. Communication proceeds securely with this key.

12*. Error message "MAC Failure" is sent, and the protocol is aborted.

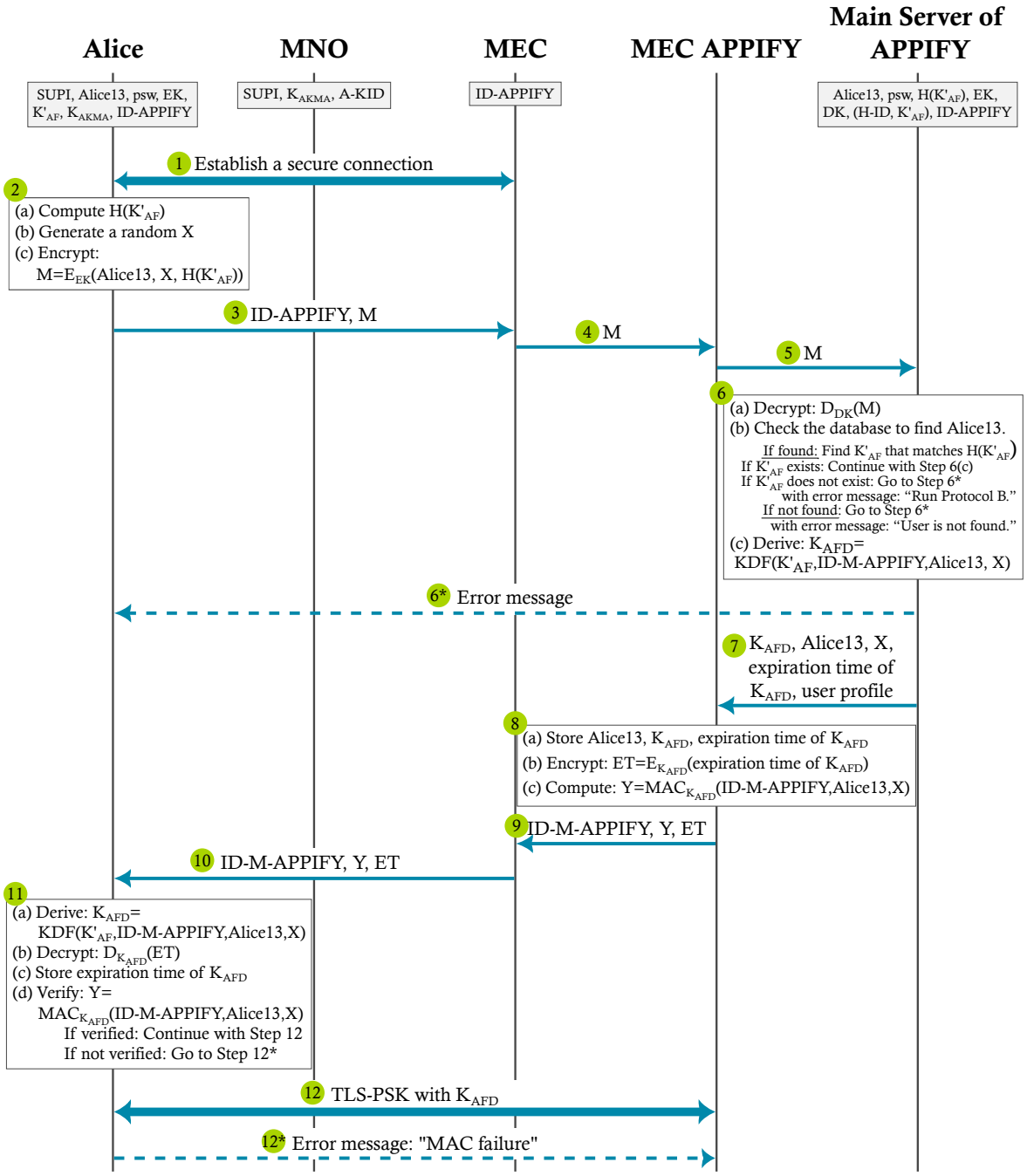


**Figure 9.** Communication flow of Protocol 5C: connecting to the application in the MEC host.

The application may have a time limit that defines how long a user remains signed in for S-APPIFY and M-APPIFY. After that time, the user is kicked out, which means the TLS connection is dropped.

Alice may want to re-establish a connection with the same M-APPIFY after some time, assuming the time limit is not over. In this case, the following steps are followed.

Is the TLS connection still on?

Yes: Protocol 5C continues with Step 12.

No: Is $K_{AFD}$ is still valid?

Yes: TLS session resumption [54] is run with the key $K_{AFD}$, and Protocol 5C continues with Step 12.

No: Alice needs to start Protocol 5C from Step 1. If Alice does not have $K'_{AF}$ stored, she first needs to run Protocol 5B to renew the key $K'_{AF}$.

### *5.4. Protocol 5D: Changing the MEC Host*

Protocol 5D is run when Alice, who is actively using M-APPIFY (this means Alice and M-APPIFY are in Step 12 of Protocol 5C), is also moving. Let us recall the case where Alice travels from one city to another in the same country, say from Helsinki to Tampere. While she moves, the MEC host that provides services to Alice changes. During this trip, MNO might or might not change. We focus on the case where MNO does not change and leave the extension to the roaming case for future work.

---

**Protocol 5 D:** Changing to MEC APPIFY in another MEC host

---

Goal. Transferring the connection with MEC APPIFY in one MEC host to MEC APPIFY in another MEC Host through the main server in a secure and privacy-preserving way.

*The protocol:*

**Ongoing secure communication with M-APPIFY:**

0. The protocol starts when there is an existing connection between Alice and source M-APPIFY resulting from Protocol 5C, i.e., Step 12 of Protocol 5C.

**Request for MEC transfer:**

1. - When MEO determines the target MEC host, MEO provides this information to the source MEC host.
   - Source MEC notifies source M-APPIFY that the connection of Alice13 is being transferred to another MEC host and M-APPIFY.
2. Source M-APPIFY sends Alice13, $K_{AFD}$, and the target MEC host information to S-APPIFY. Source M-APPIFY also sends the service state information of Alice13. This state information was already mentioned in Section 1.

**Distribution of session credentials:**

3. (a) S-APPIFY generates a random R.
   (b) S-APPIFY encrypts R with $K'_{AF}$: $E_{K'_{AF}}(R)$.
   (c) S-APPIFY derives the key $K'_{AFD}$=KDF($K'_{AF}$, ID-M-APPIFY-T, Alice13, R) where ID-M-APPIFY-T is the identifier of the target M-APPIFY.
4. S-APPIFY sends $K'_{AFD}$, Alice13, R, user profile, service state information of Alice13, and the expiration time of $K'_{AFD}$ to the target M-APPIFY.
5. (a) Target M-APPIFY stores Alice13, $K'_{AFD}$, R, user profile, service state information of Alice13, and the expiration time of $K'_{AFD}$.
   (b) Target M-APPIFY encrypts ET'=$E_{K'_{AFD}}$(expiration time of $K'_{AFD}$).
6. Target M-APPIFY sends an acknowledgment to S-APPIFY that it received the message.
7. S-APPIFY sends $E_{K'_{AF}}$(R), ID-M-APPIFY-T, and ET' to the source M-APPIFY.
8. Source M-APPIFY sends $E_{K'_{AF}}$(R), ID-M-APPIFY-T, ET', and the target MEC host information to Alice.
9. (a) Alice decrypts $E_{K'_{AF}}$(R) with $K'_{AF}$ to retrieve R.
   (b) Alice derives $K'_{AFD}$=KDF($K'_{AF}$, ID-M-APPIFY-T, Alice13, R).
   (c) Alice then decrypts $D_{K'_{AFD}}$(ET').
   (d) Alice stores the expiration time of $K'_{AFD}$.

**Secure communication with target MEC:**

10. Alice establishes a secure connection with the target MEC host, e.g., a DTLS connection.

**Secure communication with target M-APPIFY:**

11. Alice sends R with ID-APPIFY to the target MEC host.
12. Target MEC host forwards R to the target M-APPIFY.
13. Target M-APPIFY checks the database for R and finds related data as Alice13, $K'_{AFD}$, and the user information.
14. Target M-APPIFY initiates a TLS-PSK connection with Alice with the key $K'_{AFD}$.

---

There may be no available MEC host with M-APPIFY near where the user is moving to. In this case, Alice continues to be served through S-APPIFY, and Protocol 5B is run.

This can be enhanced by an option where the source M-APPIFY informs the main server about losing connection to Alice13. We do not explain this option further and leave it for future work.

When Alice is moving and she is about to leave the service area of the MEC host, MEC orchestrator (MEO) finds the target MEC host and the target M-APPIFY [6]. Protocol 5D is explained step by step below, and the communication flow is shown in Figure 10.

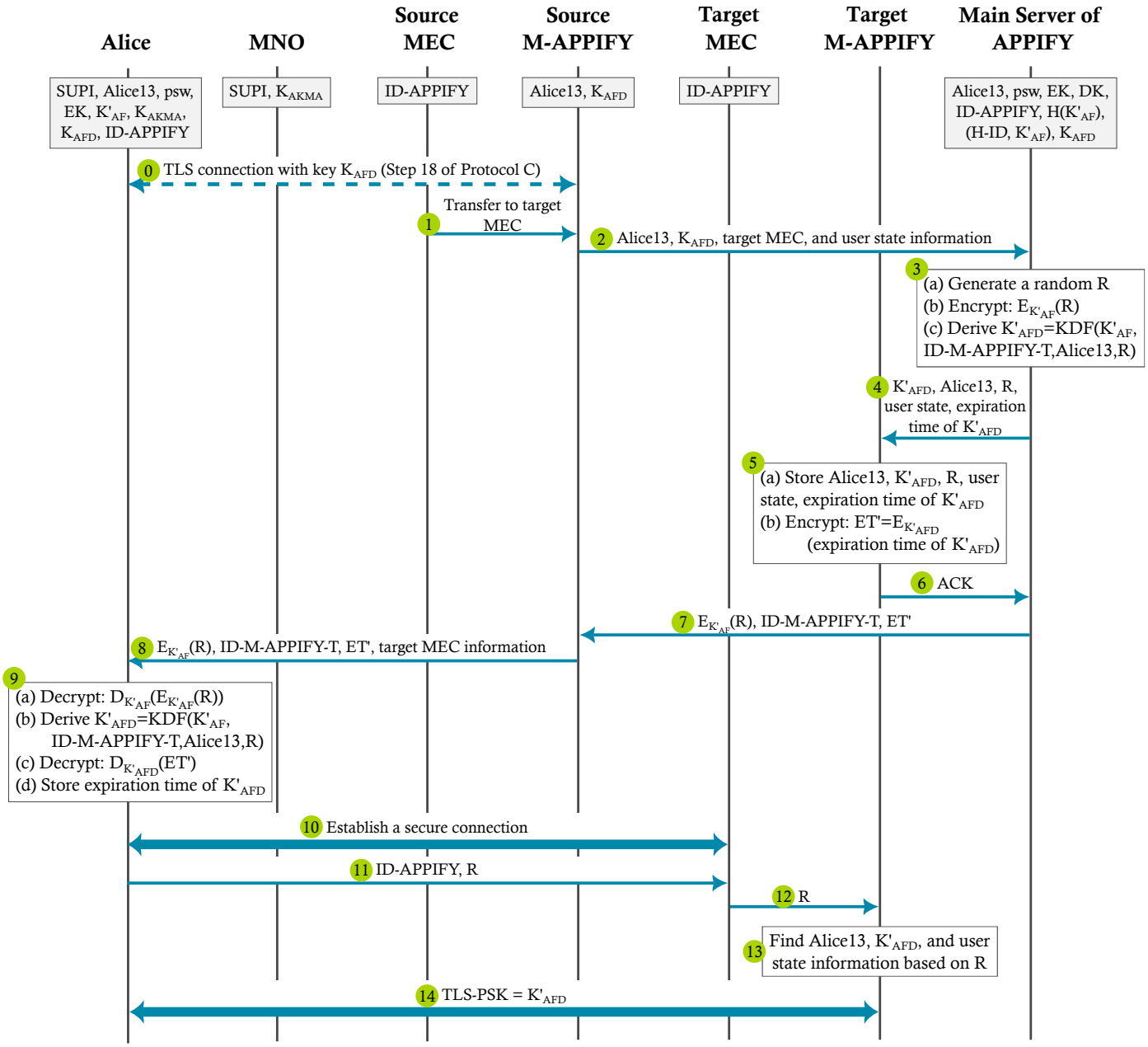


**Figure 10.** Communication flow of Protocol 5D: changing the MEC host.

When the TLS connection between Alice and target M-APPIFY is established successfully, target M-APPIFY continues providing the services to Alice instead of the source M-APPIFY. Note that until this happens, Alice has been receiving services from the source M-APPIFY. The TLS connection with source M-APPIFY is terminated when the TLS connection with target M-APPIFY is established. If the connection with source M-APPIFY is lost before the connection with target M-APPIFY is established, the service continues with the main server.

## 6. Analysis

In this section, we provide a feasibility, security, and privacy analysis of the solution that is presented in Section 5.

### *6.1. Feasibility Analysis*

Our solution uses the privacy-enhanced version of 3GPP AKMA for the authentication and key sharing between the user Alice and the main server of the application S-APPIFY in Protocol 5A (Section 5.1) and Protocol 5B (Section 5.2). The shared key resulting from AKMA is used later in Protocol 5C (Section 5.3) and Protocol 5D (Section 5.4) when Alice wants to obtain service from the M-APPIFY.

It makes sense to utilize the AKMA service in our setting because AKMA is created in 5G to support applications, and APPIFY is one such application. MNO already verifies the identity of the user during the subscription and already authenticates the user during the network access. Therefore, AKMA is using the capability of MNO to help the user and S-APPIFY authenticate each other. In addition, AKMA is a better option for authentication, and key sharing between Alice and S-APPIFY compared to, e.g., a solution where S-APPIFY creates a random key and sends it back to Alice. If one of the parties loses the shared key, the recovery of the keys between these parties requires less effort with AKMA.

Our solution works well in the situation when the user having a single user_id is using several devices, each with a different SUPI. Then, there would be an (H-ID, $K'_{AF}$) pair corresponding to each device, associated with that user_id in S-APPIFY. When the user runs Protocol 5C, M-APPIFY receives the user profile and continues serving the user with the correct device.

In Section 4.9, we showed that PE-AKMA is a feasible solution and that adapting it to AKMA TLS Profiles 1 and 2, defined in 3GPP TS 33.535 [14], is also feasible.

Protocols 5A and 5B are based on the privacy-enhanced Profile 1, described in Section 4.8.1. The communication overhead of Protocols 5A and 5B, compared to PE-AKMA Profile 1, is one extra message with a length smaller than 0.5 KB. We also included key confirmation for Alice to the protocols. Since this key confirmation is embedded in the success messages, the message length in Step 9 increases, but the increment is less than 1 KB. Computational overhead includes one database search, two hash functions in S-APPIFY, and one hash function for Alice.

Protocol 5C is based on the MEC architecture defined in [6]. For communication cost, the additional messages over those in [6] are for secure channel establishments, e.g., DTLS and TLS, in Steps 1 and 12 in Protocol 5C. In order to implement privacy protection, Protocol 5C includes an additional public key and symmetric key operations and hash functions. The UE computes three hash functions, one public key operation, and one symmetric key operation. The M-APPIFY computes one hash function and one symmetric key operation, whereas S-APPIFY does one database search and computes one hash function and one public key operation. Note that MNO and MEC do not have any additional computational load due to the run of Protocol 5C.

ETSI GR MEC 021 [7] defines three ways to transfer user context between the source and target M-APPIFYs. We chose to move the user context via the main server of the MEC application. Similar to Protocol 5C, the number of messages in Protocol 5D does not change compared to the solution in [7], other than the messages needed to establish DTLS/TLS connections. The following is the computational overhead in Protocol 5D. The UE computes one hash function and two symmetric key operations; the target M-APPIFY computes one symmetric key operation; S-APPIFY computes one hash function and one symmetric key operation.

Overall, the privacy improvements require somewhat more storage and computation. DTLS and TLS connections constitute the communication overheads to obtain privacy. These protocols are based on state-of-the-art systems, and the functions used in the protocols are standardized.

In summary, our design uses well-known techniques to improve the privacy of MEC users in the 5G system. We conclude that it is feasible to implement our solution.

### *6.2. Security Analysis*

It is natural to use usernames and passwords for accessing the applications. However, when Alice uses PE-AKMA in Protocols 5A and 5B, she does not need to enter her password for the application every time she uses the application. This is convenient and also decreases the chance of the password being compromised.

In our solution, secure communication channels are established between the parties, which protects the integrity and confidentiality of the messages sent. The channel between Alice and MNO is secured after the Authentication and Key Agreement (AKA) is run.

In Protocols 5A and 5B, the AKMA profile 1 with PE-AKMA for TLS-based protocols (Section 4.8.1) is used. Consequently, a TLS connection is established between Alice and the main server of APPIFY before she sends her identifiers and password. Therefore, Alice can be sure that she is talking to the correct server, and that these identifiers and passwords are not visible to outsiders or MNO.

Protocols 5C and 5D focus on the usage of APPIFY in MEC hosts. Therefore, there should also be a secure connection between Alice and MEC host, e.g., DTLS is used for this purpose. The handshake can be based on a pre-shared key (resulting from AKMA) or a certificate. Moreover, 3GPP defines the usage of AKMA in the technical documents [10]. We do not discuss how to secure the connection between the user and MEC host further.

Studies in 3GPP focus on the authentication between Alice and MEC host, and between Alice and MEC system. In this paper, we focus on authentication at the application level. We use privacy-enhanced AKMA to provide mutual authentication between the user and the main server of the application. As a product of PE-AKMA, the user and the main server of the application share a key. Another application-specific key, derived from the shared key, is then used to secure the connection between the user and the MEC application, e.g., TLS-PSK in Protocols 5C and 5D. This newly derived key depends on the MEC application. Therefore, whenever the user wants to connect to a MEC application in a different MEC host, a new key can be derived from the shared AKMA key without needing to run AKMA again. In addition, since the same key is not used with different MEC hosts, the key is less likely to be compromised.

### *6.3. Privacy Analysis*

The TLS connection between Alice and M-APPIFY protects the communication from outsiders and honest-but-curious entities of the system model, e.g., MNO and MEC host. These honest-but-curious entities aim to learn as much as possible without interfering with the communication protocol [41]. In order to preserve the privacy of the user, MNO, MEC, and APPIFY should not learn anything related to the user that is not needed for providing services.

The privacy of the user is preserved against MNO with our solution. MNO learns that Alice uses S-APPIFY since PE-AKMA is run between these two entities. However, MNO cannot know whether or when Alice is using M-APPIFY.

MNO cannot learn the identifier of Alice for APPIFY, Alice13, or her password, because these are sent to S-APPIFY in Protocols 5A and 5B after the TLS connection is established. Similarly, in Protocols 5C and 5D, Alice first establishes a secure connection with the MEC host and then sends her identifier, which is encrypted with the public key of the main server. This latter encryption protects the privacy of the identifier against honest-but-curious MEC hosts.

In addition, the communication between Alice and S-APPIFY continues over the TLS channel in Protocols 5A and 5B, so MNO cannot see the data sent between the two parties. In Protocols 5C and 5D, the secure channel between Alice and MEC and the TLS channel between Alice and M-APPIFY prevents MNO from seeing the content sent between Alice and M-APPIFY.

Regarding the TLS channel between Alice and M-APPIFY in Protocols 5C and 5D, we introduced a derived key $K_{AFD}$ instead of $K'_{AF}$ as a pre-shared key. Since MNO knows the key $K'_{AF}$ but not the derived one, MNO would not be able to decrypt the communication channel. Therefore, MNO cannot learn the content of communication of Alice with either S-APPIFY or M-APPIFY.

Deriving $K_{AFD}$ helps to use several M-APPIFYs without the need to repeat AKMA with S-APPIFY for each connection. The derived key is relevant because using the same $K'_{AF}$ itself to secure communications with different M-APPIFYs is not a recommended practice.

The privacy of the user is also preserved against MEC. Alice sends her identifier Alice13 after encrypting it with the public key of the main server. Moreover, the TLS connection between Alice and M-APPIFY prevents MEC from learning the content of the messages between Alice and M-APPIFY. Therefore, MEC cannot learn identifiers of Alice or the details of the services she obtains from M-APPIFY.

Finally, the privacy of the user is preserved against APPIFY (both S-APPIFY and M-APPIFY). The privacy enhancements that we propose to AKMA protect the identity of the user from S-APPIFY in Protocols 5A and 5B. In Protocols 5C and 5D, Alice does not use her identifier for MNO because she shared the key and agreed on her application-specific identifier with S-APPIFY.

## 7. Final Remarks

We started with creating a secure and privacy-preserving connection between a mobile user and the MEC application in the local MEC host. The scope is restricted to the non-roaming case, where the mobile user may change MEC hosts but stays in the same mobile network.

We propose a solution that is based on the 3GPP authentication and key management for applications (AKMA). First, we conducted a formal verification of AKMA using ProVerif and found a new spoofing attack on AKMA, as well as several privacy vulnerabilities in the current AKMA specification. Second, we designed and formally verified a privacy-enhanced AKMA, where the spoofing attack is prevented and privacy vulnerabilities are mitigated. Finally, we adapted the privacy-enhanced AKMA in our solution.

One limitation of our study is that AKMA still needs to be deployed. Therefore, observing the limits of AKMA in practice and experimenting with our solution are left for future work. Future work could also include the roaming case, where the source and the target MEC hosts belong to different mobile network operators.

**Author Contributions:** Conceptualization, G.A., P.G. and V.N.; supervision, P.G. and V.N.; writing—original draft preparation, G.A.; writing—review and editing, P.G. and V.N.; formal analysis, G.A. and M.T.D.; visualization, G.A. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research is a result of a funded collaboration project between the University of Helsinki and Huawei Technologies.

**Acknowledgments:** We would like to thank Mohsin Khan and Sampo Sovio for their helpful comments, Carlos de la Torre for his technical support, and the authors of [45] for clarifying their Tamarin model of AKMA.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| 3GPP | 3rd generation partnership project |
| 5G | fifth generation |
| AAnF | AKMA anchor function |
| AF | application function |
| AKA | authentication and key agreement |
| A-KID | Akma key identifier |
| AKMA | authentication and key management for applications |
| AR | augmented reality |
| A-TID | AKMA temporary UE identifier |
| AUSF | authentication server function |
| BSF | bootstrapping server function |
| B-TID | Bootstrapping Transaction Identifier |
| DEM | data encapsulation mechanism |
| DTLS | datagram transport layer security |
| EAP-PSK | pre-shared key extensible authentication protocol method |
| ECIES | elliptic curve integrated encryption scheme |
| EK | public encryption key |
| ET | expiration time |
| ETSI | European Telecommunication Standards Institute |
| FQDN | fully qualified domain name |
| GBA | generic bootstrapping architecture |
| GPSI | generic public subscription identifier |
| H-ID | hashed identifier |
| IMSI | International Mobile Subscriber Identity |
| IND-CPA | indistinguishability under chosen-plaintext attack |
| IoT | Internet of Things |
| ITS | intelligent transportation systems |
| KDF | key derivation function |
| KEM | key encapsulation mechanism |
| M2M | machine-to-machine |
| MAC | message authentication code |
| MEC | multi-access edge computing |
| MEO | MEC orchestrator |
| MNO | mobile network operator |
| MR | mixed reality |
| MSISDN | mobile subscriber international subscriber directory number |
| MTC | machine-type communication |
| NAF | network application function |
| NEF | network exposure function |
| NIST | National Institute of Standards and Technology |
| PE-AKMA | privacy-enhanced AKMA |
| PSK | pre-shared key |
| QoE | quality of experience |
| QoS | quality of service |
| SUPI | subscription permanent identifier |
| TLS | transport layer security |
| UAV | unmanned aerial vehicle |
| UDM | unified data management |
| UE | user equipment |
| UHD | ultra high definition |
| V2E | vehicle-to-everything |
| VR | virtual reality |

## Appendix A. Generic Bootstrapping Architecture (GBA)

Generic bootstrapping architecture (GBA) is a service used to authenticate the user to an application based on the AKA protocol [11]. A bootstrapping server function (BSF) and UE mutually authenticate each other by executing an AKA protocol. In the process, they also agree on session keys to be used later between UE and a network application function (NAF) [11,12]. The UE may run GBA independently of its radio access to MNO; hence, GBA also works over WiFi or a wired connection [17]. After the bootstrapping is finalized and the session keys are shared, UE and NAF run application-specific protocols in which the protection of the message is done with these session keys [11].

The bootstrapping procedure is further explained in 3GPP TS 33.220 [11]. First, UE sends a request to NAF. If NAF supports GBA, it returns a request to UE for the bootstrapping initiation. Then, UE starts bootstrapping authentication with BSF, which is done by the AKA protocol. In the end, both BSF and UE derive a shared key (Ks) and a bootstrapping transaction identifier (B-TID). The application-specific key ($Ks_{NAF}$) is derived from Ks by both BSF and UE. When bootstrapping is complete, the bootstrapping usage procedure starts. This procedure begins with UE sending B-TID to NAF. Next, NAF forwards the B-TID, along with its own identity NAF-ID, to BSF. Then, BSF replies with the application-specific key $Ks_{NAF}$ and the lifetime of the key. Once NAF obtains the shared key, it may start secure communications with UE, enabled with the shared key $Ks_{NAF}$.

## Appendix B. ProVerif

The construction of a protocol model in ProVerif requires three parts: (i) parameter declarations to formalize cryptographic primitives, (ii) process macros to define the protocol rules, and (iii) the main process to encode the protocol using the macros [15].

The declarations part includes defining the types, free names, and constructors that are used in the processes. In addition, events and queries are also defined in this part. Events are annotated to mark important stages of the protocol, i.e., key derivation, message sending, and message receiving. Moreover, ProVerif uses queries to check secrecy, reachability, and authentication properties [21]. The secrecy of some terms t, e.g., key, is checked by verifying that there is no trace where t is sent over a public channel. Authentication properties are formulated as correspondence assertions between events [15,21].

In the macros part, every protocol party is described by sub-processes, while the main process part defines how the overall protocol is executed by using sub-processes. Examples of the execution syntax are given below [15]:

- `P,Q` : denotes the processes P and Q.
- `P|Q` : the participants of the protocol run the processes P and Q in parallel.
- `!P` : the process P | P | P | ... | P, which creates an unbounded number of sessions.
- `new n:t;P` : introduces free name n with type t to the process P, used to create nonces, new identities, new keys, etc.
- `in(M,N);P` : awaits message N from channel M and then runs the process P.
- `out(M,N);P` : sends message N on channel M and then runs the process P.

## Appendix C. Key and Data Encapsulation Mechanisms (KEM/DEM)

In most practical situations, and casting aside digital signatures, public-key cryptography is used to encrypt a symmetric (shared) key $K_s$, then the *real* information is indeed encrypted utilizing $K_s$ with a symmetric cipher. A key encapsulation mechanism (KEM) is a public key scheme that is used to generate and establish a shared secret key between two parties. More precisely, a KEM is a triple of algorithms KEM=(KeyGen(), Key-Encaps(), Key-Decaps()) and a corresponding key space K, where

- The probabilistic key generation algorithm KeyGen() returns a public/secret-key pair (PK,SK).
- The probabilistic key encapsulation algorithm Key-Encaps(PK) takes as input a public key PK and outputs a ciphertext C as well as a key $K_s \in K$.

- The deterministic key decapsulation algorithm Key-Decaps(SK,C) takes as input a secret key SK and a ciphertext C and returns a key $K_s \in K$ or $\perp$ denoting failure.

Naturally, KEM is assumed to be sound, i.e., if PK and SK belong to the same pair, and Key-Encaps(PK) outputs C and $K_s$, then Key-Decaps(SK,C) outputs the same $K_s$.

In the KEM/DEM formalism [52,55,56], a KEM is given alongside a data encapsulation mechanism (DEM). The DEM ensures the secrecy and integrity of data using symmetric-key primitives. A DEM is given by two algorithms, DEM=(Data-Encaps,Data-Decaps)), where

- The deterministic data encapsulation algorithm Data-Encaps($K_s$, M) takes as input a symmetric key $K_s \in K$, a plaintext M, and outputs a ciphertext N.
- The deterministic data decapsulation algorithm Data-Decaps($K_s$, N) takes as input a symmetric key $K_s \in K$, a ciphertext N, and returns a plaintext M.

The DEM is also assumed to be sound, i.e., Data-Decaps($K_s$,Data-Encaps($K_s$, M))=M.

An example of a widely used KEM/DEM is the elliptic curve integrated encryption scheme (ECIES), which is used to encrypt the SUPI in 5G AKA (Annex C of [48]).

It is worth mentioning that at the time of writing, the National Institute of Standards and Technology in the USA (NIST) is in the process of standardizing quantum-resistant KEMs, which are expected to replace *classical* KEMs (including ECIES) in the near future. Thus, we chose to use general KEMs instead of the usual public-key encryption to allow our enhanced protocol to provide more flexibility.

## Appendix D. TLS with Pre-Shared Key

The pre-shared key (PSK) was specified for TLS in 2005 and is a part of TLS 1.3 [57]. PSK is a shared secret between two parties, previously shared through a secret channel before it needs to be used [58].

PSK can be established on both sides in a TLS handshake when the pre_shared_key extension is used. The server provides a PSK identity and a unique key to the client via the NewSessionTicket message. This PSK can then be used in another handshake to establish a new TLS connection, called session resumption with a PSK [54,58]. If the connection is established with PSK, then this new connection is cryptographically tied to the initial connection where the PSK itself was established [58].

PSK can also be shared externally. In this case, the PSK identity should be provisioned as well [57,58]. The use cases of external PSK include certificateless server-to-server communication and devices with limited computational capabilities. Machine-to-machine communication may prefer to use external PSKs to establish the TLS connection, thus avoiding the management of PKI certificates and the overhead of provisioning [57].

PSK extension includes a label to identify the key and ticket age for the PSKs that are established during the TLS handshake. If the external PSK is used, the ticket age is ignored. When the PSK is used to authenticate the client, the server must not send the certificate request or certificate to verify messages. If necessary, the certificate can be sent after the main handshake is completed during post-handshake authentication [54].

## Appendix E. Formal Verification Model of AKMA

This section presents the formal verification of the AKMA, described in Section 4.1, by using ProVerif.

### *Appendix E.1. Constructing the Protocol*

Next, we explain how we construct the model of the AKMA protocol in ProVerif.

#### Appendix E.1.1. Declarations

The channels have free names that are public, so the attacker knows them. If these channels and free names are declared private (by adding [private] to the end), then they cannot be learned by the attacker.

- **types:** We only use the type `key` from the built-in types.
- **channels:** We defined free private channels`chUEMNO`, `chMNOAF` for the channels between UE and MNO, as well as between MNO and AF, respectively. Another channel is the public channel `chUEAF` between UE and AF.
- **identifiers and keys:** We defined the secret identifier `SUPI`, secret key `K`, and the public identifiers for the MNO and AF: `ID-MNO, ID-AF`.
- **functions:** List of functions include key derivation functions, i.e., `fun fKAKMA(bitstring,key):key` and `fun fKAF(bitstring,key):key`, and encryption-decryption functions, i.e., `fun senckey(bitstring,key):bitstring` and `reduc forall x:bitstring, y:key; sdeckey(senckey(x,y),y)=x`.

Appendix E.1.2. Process Macros

There are three parties in the AKMA protocol. Therefore, we prepared three sub-processes, one for each party: UE, MNO, and AF.

We implemented the protocol explained in Section 4.1, and the step numbers in the code correspond to the numbers in Figure 3.

Appendix E.1.3. Main Process

The main process part of our ProVerif model of AKMA is defined as follows:

```
process
     new ID_MNO:bitstring;
     new ID_AF:bitstring;
     (!UE(SUPI,ID_MNO,ID_AF,K) | !MNO(SUPI,ID_MNO,K) | !AF(ID_AF))
```

In every session, new identifiers, ID_MNO and ID_AF, are derived. Then, the participants, UE, MNO, and AF, run their processes in parallel by creating an unbounded number of sessions.

*Appendix E.2. Results*

We now present how we constructed the queries in our ProVerif model of AKMA. We also explain how the output of ProVerif can be analyzed. The interpretation of these outputs can be found in Section 4.2.3.

- **Secrecy:** The `query attacker(M)` verifies whether the attacker can capture the M, where M can be a key, identifier, or message. However, in order to define this query, the M should be declared at the beginning of the code. Therefore, this query cannot test if the attacker captures the keys or identifiers that are created or derived during the process.

To test the secrecy of the identifier and keys, A-KID, $K_{AKMA}$, and $K_{AF}$, we defined private nonces, nAKID, nKAKMA, and nKAF, respectively, in the declaration part. Then, we use symmetric encryption (functions senc and senckey): `senc(nAKID,A_KID)`, `senckey(nKAKMA,KAKMA)`, and `senckey(nKAF,KAF)`. At the end of the sub-process of UE, these encrypted messages are sent to the public channel. This way, if any of the private nAKID, nKAKMA, and nKAF are known by the attacker, this shows that corresponding A-KID, $K_{AKMA}$, and $K_{AF}$, respectively, must also be known by the attacker [15].

We check the secrecy of the following identifiers and keys: K, SUPI, A-KID, $K_{AKMA}$, and $K_{AF}$.

ProVerif provides the following output:

```
Verification summary:
Query not attacker(K[]) is true.
Query not attacker(SUPI[]) is true.
Query not attacker(nAKID[]) is false.
Query not attacker(nKAKMA[]) is true.
Query not attacker(nKAF[]) is true.
```

The ProVerif manual [15] explains that the test result of the query attacker as `RESULT not attacker:(test[]) is true`, meaning that the attacker failed to capture the free name `test`. On the other hand, if the attacker has obtained the free name `test`, the result is denoted by the `RESULT not attacker:(test[]) is false`.

- **Strong secrecy:** We check the strong secrecy of SUPI and K.

  ProVerif provides the following outputs:

```
Verification summary:
Non-interference SUPI is true.
Non-interference K is true.
```

- **Weak Secrecy:** We check the weak secrecy of SUPI and K.

  ProVerif provides the following outputs:

```
Verification summary:
Weak secret SUPI is true.
Weak secret K is true.
```

- **Forward Secrecy:** In order to prove forward secrecy, we updated the main process of the protocol as follows.

```
process
     new ID_MNO:bitstring;
     new ID_AF:bitstring;
     (!UE(SUPI,ID_MNO,ID_AF,K) | !MNO(SUPI,ID_MNO,K) | !AF(ID_AF)| phase 1;
     out(chUEAF,( K, SUPI, ID_AF)))
```

Then, we run the same secrecy queries mentioned above, and ProVerif provides the following outputs:

```
Verification summary:
Query not attacker_p1(K[]) is false.
Query not attacker_p1(SUPI[]) is false.
Query not attacker_p1(nAKID[]) is false.
Query not attacker_p1(nKAKMA[]) is false.
Query not attacker_p1(nKAF[]) is false.
```

- **Aliveness:** We construct four queries for proving the aliveness, (1) UE with AF, (2) AF with UE, (3) MNO with AF, (4) AF with MNO. To prove the aliveness of a responder *R* with respect to a protocol initiator *I*, we show that the following assertion holds: "The agent I finished the protocol" implies that either "agent R sent its last message", or "R responded to a message (not necessarily I)."

  ProVerif provides the following outputs:

```
Verification summary:
Query event(UErecResAF(a)) ==> event(AFsendReqMNO(c,d)) ||
event(AFsendResUE(e)) is false.
Query event(AFsendResUE(a)) ==> event(UEsendReqAF(b)) is true.
Query event(MNOsendResAF(a,b,c)) ==> event(AFsendResUE(d)) ||
event(AFsendReqMNO(e,f)) is true.
Query event(AFsendResUE(a)) ==> event(MNOsendResAF(b,c,d)) is true.
```

- **Weak Agreement:** We construct four queries for proving the weak agreement, (1) UE with AF, (2) AF with UE, (3) MNO with AF, (4) AF with MNO. We constructed these queries so that if the event on the left happens, the event on the right must have happened before.

  ProVerif provides the following outputs:

```
Verification summary:
Query event(UErecResAF(a)) ==> event(AFsendResUE(b)) is false.
Query event(AFrecReqUE(a)) ==> event(UEsendReqAF(b)) is false.
Query event(MNOrecReqAF(a,b)) ==> event(AFsendReqMNO(c,d)) is true.
Query event(AFrecResMNO(a,b,c)) ==> event(MNOsendResAF(d,e,f)) is true.
```

- **Non-injective and injective agreements:** We construct four queries for proving the non-injective and injective agreements, (1) the non-injective agreement of AF with MNO on $K_{AF}$, (2) the injective agreement of AF with MNO on $K_{AF}$, (3) the non-injective agreement of MNO with AF on A-KID, and (4) the injective agreement of MNO with AF on A-KID.

Please note that we do not check any non-injective or injective agreements between UE and AF since we already proved they do not have weak agreements.

ProVerif provides the following outputs:

```
Verification summary:
Query event(AFendAKID(a)) ==> event(MNOhasAKID(c)) is true.
Query inj-event(AFendAKID(a)) ==> inj-event(MNOhasAKID(c)) cannot be proved.

Query event(AFendKAF(b)) ==> event(MNOhasKAF(d)) is true.
Query inj-event(AFendKAF(b)) ==> inj-event(MNOhasKAF(d)) cannot be proved.

Query event(MNOendAKID(a)) ==> event(AFhasAKID(c)) is true.
Query inj-event(MNOendAKID(a)) ==> inj-event(AFhasAKID(c)) cannot be proved.

Query event(MNOendKAF(b)) ==> event(AFhasKAF(d)) is false.
Query inj-event(MNOendKAF(b)) ==> inj-event(AFhasKAF(d)) is false.
```

The ProVerif manual [15] describes the results as follows: if the query returns true, there is no attack. Similarly, if the query returns false, ProVerif discovered an attack against this security property. If ProVerif returns that the query "cannot be proved", it means that ProVerif cannot prove that it is true and it cannot find an attack that proves it false. This outcome cannot be avoided because the problem of verifying protocols with a potentially unbounded number of sessions is undecidable.

## Appendix F. Formal Verification Model of PE-AKMA

This section presents the formal verification of the privacy-enhanced AKMA, described in Section 4.5, by using ProVerif.

### *Appendix F.1. Constructing the Protocol*

The construction of the PE-AKMA protocol in ProVerif is similar to what we explained in Section 4.2, except for the following:

- UE and MNO have additional private keys, i.e., `K_MNO`,`K_UE`, and MNO has public and secret (KEM) key pairs, `SK_MNO`,`PK_MNO`.
- We added other functions, e.g., to compute keyed hashed function `fun HMAC(bitstring,key):bitstring`, derive public key `fun pk(skey):pkey` from a secret key `skey`, and execute Diffie-Hellman key exchange `fun DH(pkey,skey):key`.

- We implemented the Key and Data Encapsulation Mechanisms (KEM/DEM), explained in Appendix C, in the PE-AKMA. The encapsulation algorithm consists of three functions, namely, `fun Encaps(pkey,bitstring):bitstring` to generate a randomized input for producing the KEM cipher and key, which is to provide resistance to replay attacks, `funKEMkey(bitstring):key` to generate the KEM shared key, `fun KEMCipher(bitstring):bitstring` to compute the KEM ciphertext. The KEM decapsulation function `fun DecapsKey(skey,bitstring):key`, which derives the KEM shared key from the KEM ciphertext and the KEM secret key. Finally, the encapsulation and decapsulation functions are defined by the following equation, `equation forall sk:skey,r:bitstring; DecapsKey(sk,KEMCipher(Encaps(pk(sk),r)))=KEMkey(Encaps(pk(sk),r))`.
- For Data Encapsulation Mechanism (DEM), we choose to use a macced symmetric encryption (Encrypt-then-Mac), as in the ECIES DEM used by 3GPP to encrypt the SUPI. More precisely, to encrypt a message m using $K_{sh}$, the shared key resulting from KEM, the key $K_{sh}$ is split into two parts, $K_1$ and $K_2$, then DEM output is $N = N_1 | N_2$, where $N_1$ = senckey(m, $K_1$) and $N_2$ = HMAC($N_1$, $K_2$). Similarly, the data decapsulation algorithm Data-Decaps($K_{sh}$, N), first, parses the key $K_{sh}$ and the cipher N to ($K_1$, $K_2$) and ($N_1$, $N_2$), respectively. Then, it checks the MAC using HMAC($N_2$, $K_2$). If the MAC check passes, then the algorithm proceeds to decrypt $N_1$ using $K_1$. For our ProVerif code, we use the shared key $K_{sh}$ for both HMAC and the symmetric encryption and this does not affect our proofs.
- We also updated the main process according to the updates in the process macro:

```
process
     new ID_MNO:bitstring;
     new ID_AF:bitstring;
     let PK_MNO = pk(SK_MNO) in
     out(internet, PK_MNO);
     (!UE(SUPI,ID_MNO,ID_AF,K,user_id,psw,PK_MNO,K_UE) |
     !MNO(SUPI,ID_MNO,K,SK_MNO,ID_AF) | !AF(ID_AF,ID_MNO))
```

The main difference here is the derivation of the public key of MNO and broadcasting it to the public network.

*Appendix F.2. Results*

Now we present the differences in the queries in our ProVerif model of PE-AKMA, compared to AKMA. The interpretation of these outputs can be found in Section 4.6.2.

- **Secrecy properties:** The enhancements in AKMA improved some security properties. Table 1 presents the comparison of security properties between generic AKMA and PE-AKMA.

In addition to the secrecy queries that were verified in the generic AKMA protocol, secrecy queries for $SK_{MNO}$, the KEM secret key of MNO, and the KEM shared key $K_{sh}$ are added. Please note that to avoid confusion, we denote the session key in PE-AKMA by K'_AF instead of K_AF used in AKMA.

ProVerif provides the following outputs:

```
Verification summary:
Query not attacker(SK_MNO[]) is true.
Query not attacker(nKsh[]) is true.
Query not attacker(nKpAF[]) is true.
```

- **Forward Secrecy:** We updated the main process similarly as we did earlier in Section 4.2. This time the ProVerif output is as follows:

```
Verification summary:
Query not attacker_p1(K[]) is false.
Query not attacker_p1(SUPI[]) is false.
Query not attacker_p1(nAKID[]) is false.
Query not attacker_p1(nKAKMA[]) is false.
Query not attacker_p1(SK_MNO[]) is true.
Query not attacker_p1(nKsh[]) is true.
Query not attacker_p1(nKpAF[]) is true.
```

- **Aliveness:** There is aliveness, between AF and UE, between MNO and AF, and between AF and MNO. However, the aliveness property does not hold between UE and AF.
- **Weak Agreement:** The weak agreement between UE and AF does not hold. On the other hand, we can prove that MNO has a weak agreement with AF and AF has a weak agreement with MNO.
- **Non-injective and Injective Agreements:** We introduced new keys and identifiers. Therefore, we have new queries for non-injective and injective agreements.

  ProVerif provides the following outputs:

```
Verification summary:
Query event(AFendN(a)) ==> event(MNOhasN(c)) is true.
Query inj-event(AFendN(a)) ==> inj-event(MNOhasN(c)) cannot be proved.

Query event(AFendKpAF(b)) ==> event(MNOhasKpAF(d)) is true.
Query inj-event(AFendKpAF(b)) ==> inj-event(MNOhasKpAF(d)) cannot be proved.

Query event(MNOendN(a)) ==> event(AFhasN(c)) is true.
Query inj-event(MNOendN(a)) ==> inj-event(AFhasN(c)) cannot be proved.

Query event(MNOendKpAF(b)) ==> event(AFhasKpAF(d)) is false.
Query inj-event(MNOendKpAF(b)) ==> inj-event(AFhasKpAF(d)) is false.
```

## References


1. Liu, Y.; Peng, M.; Shou, G.; Chen, Y.; Chen, S. Toward Edge Intelligence: Multiaccess Edge Computing for 5G and Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 6722–6747. [CrossRef]
2. Ranaweera, P.; Jurcut, A.; Liyanage, M. MEC-enabled 5G Use Cases: A Survey on Security Vulnerabilities and Countermeasures. *ACM Comput. Surv.* **2022**, *54*, 1–37. [CrossRef]
3. Ranaweera, P.; Jurcut, A.D.; Liyanage, M. Survey on Multi-Access Edge Computing Security and Privacy. *IEEE Commun. Surv. Tutorials* **2021**, *23*, 1078–1124. [CrossRef]
4. Liyanage, M.; Porambage, P.; Ding, A.Y.; Kalla, A. Driving forces for Multi-Access Edge Computing (MEC) IoT integration in 5G. *ICT Express* **2021**, *7*, 127–137. [CrossRef]
5. Ali, B.; Gregory, M.A.; Li, S. Multi-Access Edge Computing Architecture, Data Security and Privacy: A Review. *IEEE Access* **2021**, *9*, 18706–18721. [CrossRef]
6. ETSI. Framework and Reference Architecture. Group Specification GS MEC 003 V3.1.1, ETSI. 2022. Available online: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/03.01.01_60/gs_MEC003v030101p.pdf (accessed on 14 December 2022).
7. ETSI. Application Mobility Service API. Group Specification GS MEC 021 V2.2.1, ETSI. 2022. Available online: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/021/02.02.01_60/gs_mec021v020201p.pdf (accessed on 14 December 2022).
8. Sabella, D. MEC Federation and Mobility Aspects. In *Multi-Access Edge Computing: Software Development at the Network Edge*; Springer International Publishing: Cham, Switzerland, 2021; pp. 245–279. [CrossRef]
9. Tabatabaee Malazi, H.; Chaudhry, S.R.; Kazmi, A.; Palade, A.; Cabrera, C.; White, G.; Clarke, S. Dynamic Service Placement in Multi-Access Edge Computing: A Systematic Literature Review. *IEEE Access* **2022**, *10*, 32639–32688. [CrossRef]
10. 3GPP. Study on Security Aspects of Enhancement of Support for Edge Computing in the 5G Core (5GC). Technical Report TR 33.839 V17.1.0, 3GPP. 2022. Available online: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3759 (accessed on 14 December 2022).

11. 3GPP. Generic Authentication Architecture (GAA); Generic Bootstrapping Architecture (GBA). Technical Specification TS 33.220 V17.3.0, 3GPP. 2022. Available online: https://www.etsi.org/deliver/etsi_ts/133200_133299/133220/17.03.00_60/ts_133220v170300p.pdf (accessed on 14 December 2022).
12. Holtmanns, S.; Niemi, V.; Ginzboorg, P.; Laitinen, P.; Asokan, N. *Cellular Authentication for Mobile and Internet Services*; Wiley: Chichester, UK, 2008.
13. Ogbodo, E.U.; Abu-Mahfouz, A.M.; Kurien, A.M. A Survey on 5G and LPWAN-IoT for Improved Smart Cities and Remote Area Applications: From the Aspect of Architecture and Security. *Sensors* **2022**, *22*, 6313. [CrossRef]
14. 3GPP. Authentication and Key Management for Applications (AKMA) Based on 3GPP Credentials in the 5G System (5GS). Technical Specification TS 33.535 V17.6.0, 3GPP. 2022. Available online: https://www.etsi.org/deliver/etsi_ts/133500_133599/133535/17.06.00_60/ts_133535v170600p.pdf (accessed on 14 December 2022).
15. Blanchet, B.; Smyth, B.; Cheval, V.; Sylvestre, M. ProVerif 2.04: Automatic Cryptographic Protocol Verifier. User Manual and Tutorial, INRIA Paris-Rocquencourt. 2021. Available online: https://bblanche.gitlabpages.inria.fr/proverif/manual.pdf (accessed on 14 December 2022).
16. Nencioni, G.; Garroppo, R.G.; Olimid, R.F. 5G Multi-access Edge Computing: Security, Dependability, and Performance. *arXiv* **2021**, arXiv:2107.13374.
17. Huang, X.; Tsiatsis, V.; Palanigounder, A.; Su, L.; Yang, B. 5G Authentication and Key Management for Applications. *IEEE Commun. Stand. Mag.* **2021**, *5*, 142–148. [CrossRef]
18. Lei, W.; Soong, A.C.K.; Jianghua, L.; Yong, W.; Classon, B.; Xiao, W.; Mazzarese, D.; Yang, Z.; Saboorian, T. 5G Security System Design for All Ages. In *5G System Design*; Springer International Publishing: Cham, Switzerland, 2021; pp. 341–390. [CrossRef]
19. Guirat, I.B.; Halpin, H. Formal verification of the W3C web authentication protocol. In Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security, Raleigh, NC, USA, 10–11 April 2018; ACM: New York, NY, USA, 2018; pp. 1–10. [CrossRef]
20. Peltonen, A.; Sasse, R.; Basin, D. A comprehensive formal analysis of 5G handover. In Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks, Abu Dhabi, United Arab Emirates, 28 June–2 July 2021; ACM: New York, NY, USA, 2021; pp. 1–12. [CrossRef]
21. Edris, E.K.K.; Aiash, M.; Loo, J. Formal Verification of Authentication and Service Authorization Protocols in 5G-Enabled Device-to-Device Communications Using ProVerif. *Electronics* **2021**, *10*, 1608. [CrossRef]
22. Basin, D.; Cremers, C.; Dreier, J.; Meier, S.; Sasse, R.; Schmidt, B. Tamarin-Prover Manual: Security Protocol Analysis in the Symbolic Model. User Manual and Tutorial. 2022. Available online: https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf (accessed on 14 December 2022).
23. Okazaki, H.; Futa, Y.; Arai, K. Suitable Symbolic Models for Cryptographic Verification of Secure Protocols in ProVerif. In Proceedings of the 2018 International Symposium on Information Theory and Its Applications (ISITA), Singapore, 28–31 October 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 326–330. [CrossRef]
24. Lowe, G. A Hierarchy of Authentication Specifications. In Proceedings of the 10th Computer Security Foundations Workshop, Rockport, MA, USA, 10–12 June 1997; IEEE Comput. Soc. Press: Piscataway, NJ, USA, 1997; pp. 31–43. [CrossRef]
25. Arai, K.; Kaneko, T. Formal Verification of Improved Numeric Comparison Protocol for Secure Simple Paring in Bluetooth Using ProVerif. In Proceedings of the 2014 International Conference on Security & Management, Reading, UK, 23–24 October 2014; CSREA Press: Las Vegas, NV, USA, 2014; pp. 255–261.
26. Singh, A.; Satapathy, S.C.; Roy, A.; Gutub, A. AI-Based Mobile Edge Computing for IoT: Applications, Challenges, and Future Scope. *Arabian J. Sci. Eng.* **2022**, *47*, 9801–9831. [CrossRef]
27. Mitsis, G.; Tsiropoulou, E.E.; Papavassiliou, S. Data Offloading in UAV-Assisted Multi-Access Edge Computing Systems: A Resource-Based Pricing and User Risk-Awareness Approach. *Sensors* **2020**, *20*, 2434. [CrossRef] [PubMed]
28. Kim, Y.; Park, J.G.; Lee, J.H. Security Threats in 5G Edge Computing Environments. In Proceedings of the 2020 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Republic of Korea, 21–23 October 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 905–907. [CrossRef]
29. Kim, H.; Cha, Y.; Kim, T.; Kim, P. A Study on the Security Threats and Privacy Policy of Intelligent Video Surveillance System Considering 5G Network Architecture. In Proceedings of the 2020 International Conference on Electronics, Information, and Communication (ICEIC), Barcelona, Spain, 19–22 January 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 1–4. [CrossRef]
30. Khan, M.; Ginzboorg, P.; Niemi, V. Privacy Preserving AKMA in 5G. In Proceedings of the 5th ACM Workshop on Security Standardisation Research Workshop–SSR'19, London, UK, 11 November 2019; ACM Press: New York, NY, USA, 2019; pp. 45–56. [CrossRef]
31. Kim, J.; Han, D.G.; You, I. Design of Secure Authentication Handover Protocol for Innovative Mobile Multimedia Services in 5G MEC Environments. *J. Internet Technol.* **2022**, *23*, 1245–1261. [CrossRef]
32. Yang, T.; Wang, S.; Zhan, B.; Zhan, N.; Li, J.; Xiang, S.; Xiang, Z.; Mao, B. Formal Analysis of 5G AKMA. In *Dependable Software Engineering. Theories, Tools, and Applications*; Qin, S., Woodcock, J., Zhang, W., Eds.; Springer International Publishing: Cham, Switzerland, 2021; Volume 1307, pp. 102–121.
33. Niewolski, W.; Nowak, T.W.; Sepczuk, M.; Kotulski, Z. Token-Based Authentication Framework for 5G MEC Mobile Networks. *Electronics* **2021**, *10*, 1724. [CrossRef]

34. Ali, A.; Lin, Y.D.; Li, C.Y.; Lai, Y.C. Transparent 3rd-Party Authentication with Application Mobility for 5G Mobile Edge Computing. In Proceedings of the 2020 European Conference on Networks and Communications (EuCNC), Dubrovnik, Croatia, 15–18 June 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 219–224. [CrossRef]
35. Ali, A.; Rahman Khan, S.; Sakib, S.; Hossain, M.S.; Lin, Y.D. Federated 3GPP Mobile Edge Computing Systems: A Transparent Proxy for Third Party Authentication With Application Mobility Support. *IEEE Access* **2022**, *10*, 35106–35119. [CrossRef]
36. Sanchez-Gomez, J.; Marin-Perez, R.; Sanchez-Iborra, R.; Zamora, M.A. MEC-based architecture for interoperable and trustworthy internet of moving things. *Digit. Commun. Netw.* **2022**. [CrossRef]
37. Zhang, P.; Durresi, M.; Durresi, A. Multi-access edge computing aided mobility for privacy protection in Internet of Things. *Computing* **2019**, *101*, 729–742. [CrossRef]
38. Rice, T.; Seppala, G.; Edgar, T.W.; Cain, D.; Choi, E. Fused Sensor Analysis and Advanced Control of Industrial Field Devices for Security: Cymbiote Multi-Source Sensor Fusion Platform. In Proceedings of the Northwest Cybersecurity Symposium, Richland, WA, USA, 8–10 April 2019; ACM: New York, NY, USA, 2019; pp. 1–8. [CrossRef]
39. Herzog, J. A computational interpretation of Dolev–Yao adversaries. *Theor. Comput. Sci.* **2005**, *340*, 57–81. [CrossRef]
40. Halpern, J.Y.; Pucella, R. Modeling Adversaries in a Logic for Security Protocol Analysis. In *Formal Aspects of Security*; Goos, G., Hartmanis, J., van Leeuwen, J., Abdallah, A.E., Ryan, P., Schneider, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2629, pp. 115–132. [CrossRef]
41. Paverd, A.; Martin, A.; Brown, I. Modelling and Automatically Analyzing Privacy Properties for Honest-but-Curious Adversaries. Technical Report. 2014. Available online: https://www.cs.ox.ac.uk/people/andrew.paverd/casper/casper-privacy-report.pdf (accessed on 14 December 2022).
42. Moradi, A.; Venkategowda, N.K.D.; Pouria Talebi, S.; Werner, S. Distributed Kalman Filtering with Privacy against Honest-but-Curious Adversaries. In Proceedings of the 2021 55th Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 31 October–3 November 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 790–794. [CrossRef]
43. 3GPP. Generic Authentication Architecture (GAA); Access to Network Application Functions Using Hypertext Transfer Protocol over Transport Layer Security (HTTPS). Technical Specification TS 33.222 V17.2.0, 3GPP. 2022. Available online: https://www.etsi.org/deliver/etsi_ts/133200_133299/133222/17.02.00_60/ts_133222v170200p.pdf (accessed on 14 December 2022).
44. Akman, G. AKMA and PE-AKMA ProVerif Implementation. 2022. Available online: https://github.com/gizem-akman/ProVerif-PE-AKMA (accessed on 14 December 2022).
45. Yang, T.; Wang, S.; Zhan, B.; Zhan, N.; Li, J.; Xiang, S.; Xiang, Z.; Mao, B. Formal Analysis of 5G Authentication and Key Management for Applications (AKMA). *J. Syst. Archit.* **2022**, *126*, 102478. [CrossRef]
46. 3GPP. System Architecture for the 5G System (5GS). Technical Specification TS 23.501 v 16.6.0, 3GPP. 2021. Available online: https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/16.06.00_60/ts_123501v160600p.pdf (accessed on 14 December 2022).
47. Khan, M.; Ginzboorg, P.; Niemi, V. AKMA: Delegated Authentication System of 5G. *IEEE Commun. Stand. Mag.* **2021**, *5*, 56–61. [CrossRef]
48. 3GPP. Security Architecture and Procedures for 5G System. Technical Specification TS 33.501 V17.5.0, 3GPP. 2022. Available online: https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/17.05.00_60/ts_133501v170500p.pdf (accessed on 14 December 2022).
49. Brisfors, M.; Forsmark, S.; Dubrova, E. How Deep Learning Helps Compromising USIM. In *Smart Card Research and Advanced Applications*; Liardet, P.Y., Mentens, N., Eds.; Springer International Publishing: Berlin/Heidelberg, Germany, 2021; Volume 12609, pp. 135–150. [CrossRef]
50. Katz, J.; Lindell, Y. *Introduction to Modern Cryptography*, 3rd ed.; Chapman & Hall/CRC Cryptography and Network Security Series; CRC Press: Boca Raton, FL, USA, 2020.
51. Fujisaki, E.; Okamoto, T. Secure Integration of Asymmetric and Symmetric Encryption Schemes. *J. Cryptol.* **2013**, *26*, 80–101. [CrossRef]
52. Shoup, V. A proposal for an ISO standard for public key encryption (version 2.1). *IACR-Print Arch.* **2001**, *112*, 56.
53. 3GPP. Security Aspects of Enhancement of Support for Enabling Edge Applications. Technical Specification TS 33.558 V0.3.0, 3GPP, 2021. Available online: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3929 (accessed on 14 December 2022).
54. Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.3. IETF RFC 8446. 2018. Available online: https://www.rfc-editor.org/info/rfc8446 (accessed on 14 December 2022).
55. Cramer, R.; Shoup, V. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology—CRYPTO '98*; Goos, G., Hartmanis, J., van Leeuwen, J., Krawczyk, H., Eds.; Springer: Berlin/Heidelberg, Germany, 1998; Volume 1462, pp. 13–25. [CrossRef]
56. Cramer, R.; Shoup, V. Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack. *SIAM J. Comput.* **2003**, *33*, 167–226. [CrossRef]

57. Housley, R.; Hoyland, J.; Sethi, M.; Wood, C.A. Guidance for External PSK Usage in TLS. Internet-Draft draft-ietf-tls-external-psk-guidance-06, IETF. 2022. Available online: https://datatracker.ietf.org/doc/html/draft-ietf-tls-external-psk-guidance-06 (accessed on 14 December 2022).
58. IBM. Session Resumption with a Pre-Shared Key. 2022. Available online: https://www.ibm.com/docs/en/sdk-java-technology/8?topic=handshake-session-resumption-pre-shared-key (accessed on 21 June 2022).