

AKMA for Secure Multi-access Edge Computing Mobility in 5G

Gizem Akman[1,2(✉)], Philip Ginzboorg[3], and Valtteri Niemi[1,2]

[1] University of Helsinki, Helsinki, Finland
{gizem.akman,valtteri.niemi}@helsinki.fi
[2] Helsinki Institute for Information Technology (HIIT), Helsinki, Finland
[3] Huawei Technologies, Helsinki, Finland
philip.ginzboorg@huawei.com

Abstract. Multi-Access Edge Computing (MEC) extends the cloud computing capabilities to the edge of the 5G network. The current 3rd Generation Partnership Project (3GPP) and European Telecommunications Standard Institute (ETSI) specifications about MEC cover connectivity between a mobile user and a MEC host, including security, but they do not extend to application-level security and privacy issues. We solve the problem of creating a secure and privacy-preserving communication channel between a mobile user and a distributed MEC application. The solution is limited to the case where the mobile network remains the same during communication. The solution is based on the 3GPP AKMA for authentication and key sharing. It includes protocols for (1) registering the user to the main server of the application, (2) updating the user information and shared keys with the main server of the application, (3) using the application in the MEC host in the static case, (4) using the application in MEC host while moving.

Keywords: MEC · 5G · AKMA · MEC mobility · Security · Privacy

1 Introduction

The Fifth Generation (5G) mobile network is developed to have reliable, energy-efficient, and high capacity service with low latency. 5G can offer reliability and delay-critical services [11]. 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 by reducing the energy consumption by 90%, and support for 300,000 devices within a single cell [11,16,17].

Internet of Things (IoT) is a growing ecosystem with massive interconnections of heterogeneous physical objects, and the number of IoT devices is increasing exponentially. Thus, these devices generate a large amount of data. However, IoT devices are resource-constrained. They have low battery power, low storage capacity, small processing power, and limited communication capacity [11,12]. Therefore, IoT applications require a decentralized local service for location

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
O. Gervasi et al. (Eds.): ICCSA 2022 Workshops, LNCS 13380, pp. 432–449, 2022.
https://doi.org/10.1007/978-3-031-10542-5_30

awareness, reliability, low latency [11], and access to centralized cloud computing facilities for data processing and storing [12].

Cloud computing-based services support IoT applications in providing computing resources and storage. However, cloud computing can cause latency issues, jitter, service interruption, and a single point of failure, which is inconvenient, especially for time-sensitive IoT services [11,17]. Multi-access Edge Computing (MEC) is developed to extend the capabilities of cloud computing to the edge of the mobile network and provide improved Quality of Service (QoS) and Quality of Experience (QoE) [6,12]. In order to do this, MEC hosts are deployed to the edge of the mobile network.

Together with 5G technology, MEC provides proximity, very low latency, high bandwidth, location awareness, real-time responsiveness, security and privacy protection, proximate data outsourcing, and mobility support [11,12,17]. 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 Vehicles (UAV), Vehicle-to-Everything (V2E), Intelligent Transportation Systems (ITS), serious gaming, factory automation, healthcare, smart grid, smart home, smart city, smart retail, and smart agriculture [11,16,17].

The application instance in the device of the user may relocate from one MEC host to another when the user moves within the network [7]. The source MEC host is the MEC host where the user is initially connected to and provided services. The target MEC host is where the user is being relocated.

MEC applications can be sensitive to User Equipment (UE) mobility or not. For example, suppose the purpose of the MEC application is to process the traffic in the local MEC host. Then, it is not sensitive to UE mobility because the service can continue rapidly after the handover without disrupting the service. However, if the service requires continuous connectivity between the device and MEC applications, the MEC application becomes sensitive to the UE mobility and requires seamless handover to another MEC host [8]. In order to provide seamless relocation, it is necessary to transfer the user context (i.e., user-specific application state) associated with the device along with the application instance between the source and target MEC hosts to continue providing services without interruption [7,19]. The user context transfer can happen in three ways: device application-assisted state transfer, MEC-assisted state transfer, and application self-controlled state transfer [7]. MEC orchestrator, which has an overview of the MEC system, selects a new host for relocation and initiates the transfer of an application instance from the source MEC host to the target MEC host [8,13].

Considering that MEC applications process data related to users, share secret keys, and store private data, relocating the services between different MEC hosts might introduce security and privacy issues [21]. Therefore, UE mobility should 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, unauthorized UE may start using the

MEC application, or the UE may share its information with the unauthorized MEC application [3].

The problem we are addressing in this paper is how to secure communication between the MEC application client in UE and the MEC application in the MEC host, both when the user is static and when the user is mobile but is still connected to the mobile network of the same Mobile Network Operator (MNO). Our contribution is a secure and privacy-friendly solution for using the MEC application by static and mobile users. The solution includes the following protocols: (A) registration of the user to the main server of the application, (B) updating the user information and shared keys with the main server of the application, (C) usage of the application in the MEC host in the static case, (D) usage of the application in MEC host in the mobile case. We use the 3rd Generation Partnership Project (3GPP) Authentication and Key Management for Applications (AKMA) service for authentication and key sharing.

We continue below with background information and related work in Sect. 2. The system model is explained in Sect. 3 and the solution that includes four protocols is presented in Sect. 4. Then, the solution is analyzed in Sect. 5. The paper ends with the conclusion and future work in Sect. 6.

2 Background and Related Work

In this section, we give additional background about the MEC framework and AKMA service and summarize the related work.

2.1 MEC Framework

European Telecommunications Standard Institute (ETSI) initiated MEC standardization in 2014 [13]. The framework of MEC consists of three levels [8]: MEC system, MEC host, and network.

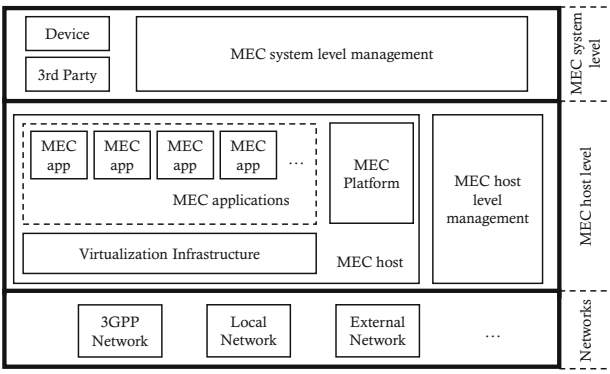

Fig. 1. MEC Framework, adapted from [8]

The MEC system level includes the MEC orchestrator and user application. Network can be the 3GPP, local, or external network. The MEC host-level has MEC host and MEC host-level management. The MEC host is deployed to the edge of the mobile network, which includes a MEC platform, MEC applications, and virtualization infrastructure. Overall MEC framework is shown in Fig. 1.

2.2 AKMA Service

Authentication and Key Management for Applications (AKMA) is a service in the 5G system to support authentication and security protection between the UE and applications. AKMA is using similar principles, e.g., those of Generic Bootstrapping Architecture (GBA) in the legacy mobile networks, while adapting to the architectural changes in the 5G system to meet the requirements for scenarios related to the Internet of Things [9]. The authentication and key agreement rely on the credentials and identities of the mobile subscriber, which are pre-established with Mobile Network Operator (MNO) for 5G access [10]. The AKMA service will help mutual authentication between UE and applications [9].

The key for AKMA, K_{AKMA}, is derived from the shared key that results from the primary authentication between the UE and the 5G mobile network. In addition, an identifier A-KID is assigned to the user to identify the key K_{AKMA}. Another application-specific key K_{AF} is derived from K_{AKMA} by UE and mobile network [10]. The key K_{AKMA} and 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, K_{AF} can still be used until the lifetime of K_{AF} expires, or the application session ends. Then, a new K_{AF} will be derived from the new K_{AKMA} [1].

2.3 Related Work

Niewolski et al. [14] propose a solution using a token-based authentication framework for the 5G MEC system. However, they do not consider the case where the user is mobile. The articles [4,5] propose third-party authentication for securing the MEC application mobility with different approaches for the state transfer of the MEC application during mobility, such as token-based and federated state transfer. The solution of Sanchez-Gomez et al. [20] uses Pre-Shared Key Extensible Authentication Protocol Method (EAP-PSK) to authenticate of the user and MEC application for both static and mobile users.

3 System Model

Our system model can be illustrated by the following scenario: 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, which could be an application, e.g., for video streaming, AR, or mobile gaming. Since Alice stays in her country, it is very likely that MNO will

not change during her trip. However, MEC coverage is only local, unlike MNO coverage. Alice would need to change several MEC hosts between Helsinki and Tampere, therefore, several MEC application servers. Nonetheless, she wants to continue using APPIFY with the benefits of MEC services, like low latency and high bandwidth. Additionally, she wants to protect her information from outsiders and the other parties in the network unless the information is needed to get the service. This information can be, e.g., the name of APPIFY, the content of her communication with APPIFY, and her identifiers.

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.

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 through the internet. In addition to this, the application server of APPIFY resides in MEC hosts. APPIFY in the MEC host is synchronized with its main server. Alice connects to the APPIFY in the MEC host and gets the service there instead of the main server for 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 Fig. 2. However, the number of MEC hosts with an APPIFY server is not limited to two. It should also be noted that there could be several other MEC application servers in MEC hosts other than APPIFY.

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

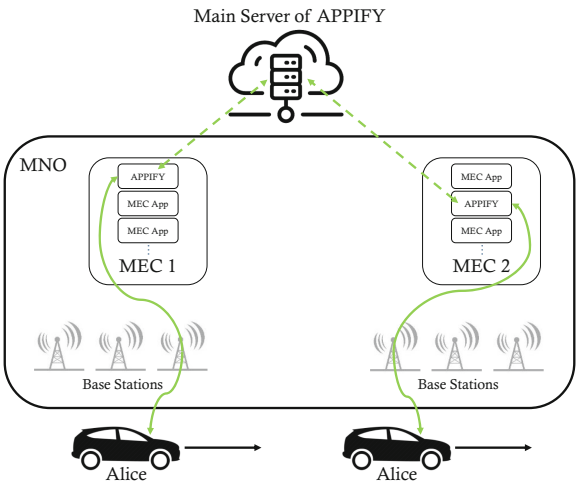

Fig. 2. System model

4 Solution

This section presents a solution that adapts AKMA for the purpose of maintaining security and privacy for the MEC applications that are used by both static and mobile users.

Alice is a subscriber of MNO, and she uses an application called APPIFY. The application APPIFY appears in the solution as the main server of APPIFY (S-APPIFY) and the APPIFY in the MEC host (M-APPIFY). Alice uses AKMA to authenticate herself to APPIFY.

S-APPIFY has public encryption and secret decryption key pair (EK, DK). The users of APPIFY use this EK to encrypt their identifiers. S-APPIFY also has a key K_{MAIN}, which is used to protect the privacy of the application identifier of the user against MNO. ID-APPIFY is the identifier of S-APPIFY and is known by MNO. The MNO has a key K_{MNO}, which is used during the generation of the user identifier that identifies Alice to S-APPIFY during AKMA. In addition, Alice has her identifier Subscription Permanent Identifier (SUPI) for MNO and identifier–password pair (Alice13, psw) for APPIFY.

Our solution consists of four protocols: A, B, C, and D. Protocols A and B are executed between Alice and the S-APPIFY. Protocol A is used to add a new user or register a new trusted device. Protocol B is for renewing EK and shared keys. This protocol is also run to establish a secure connection between Alice and S-APPIFY for the purpose of providing service to Alice from the main server. Protocol C is executed between Alice and M-APPIFY when Alice wants to start using services provided by APPIFY in the MEC host. Protocol D is used when Alice is moving and uses APPIFY in the MEC host, but the serving MEC host and M-APPIFY change.

In Protocols A and B, a Transport Layer Security (TLS) connection is established between Alice and the main server of APPIFY. Alice uses a verification key of a root certificate in the TLS connection set-up to verify the certificate of the main server. This is a well-known procedure and will not be discussed in this paper.

A secure channel between Alice and MEC host [2], e.g., a TLS or Datagram Transport Layer Security (DTLS) connection, is established in Protocols C and D. This secure channel establishment is based either on a certificate of the MEC host or, alternatively, on a pre-shared key PSK method with a key that results from AKMA between Alice and MEC [3]. This paper will not discuss further the details of the secure channel between Alice and MEC.

Next, we explain the protocols of our solution.

4.1 Protocol A: Signing Up

Protocol A 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 means a new SUPI. Protocol A runs between Alice and S-APPIFY through MNO. The protocol is explained step by step below, and the communication flow is shown in Fig. 3.

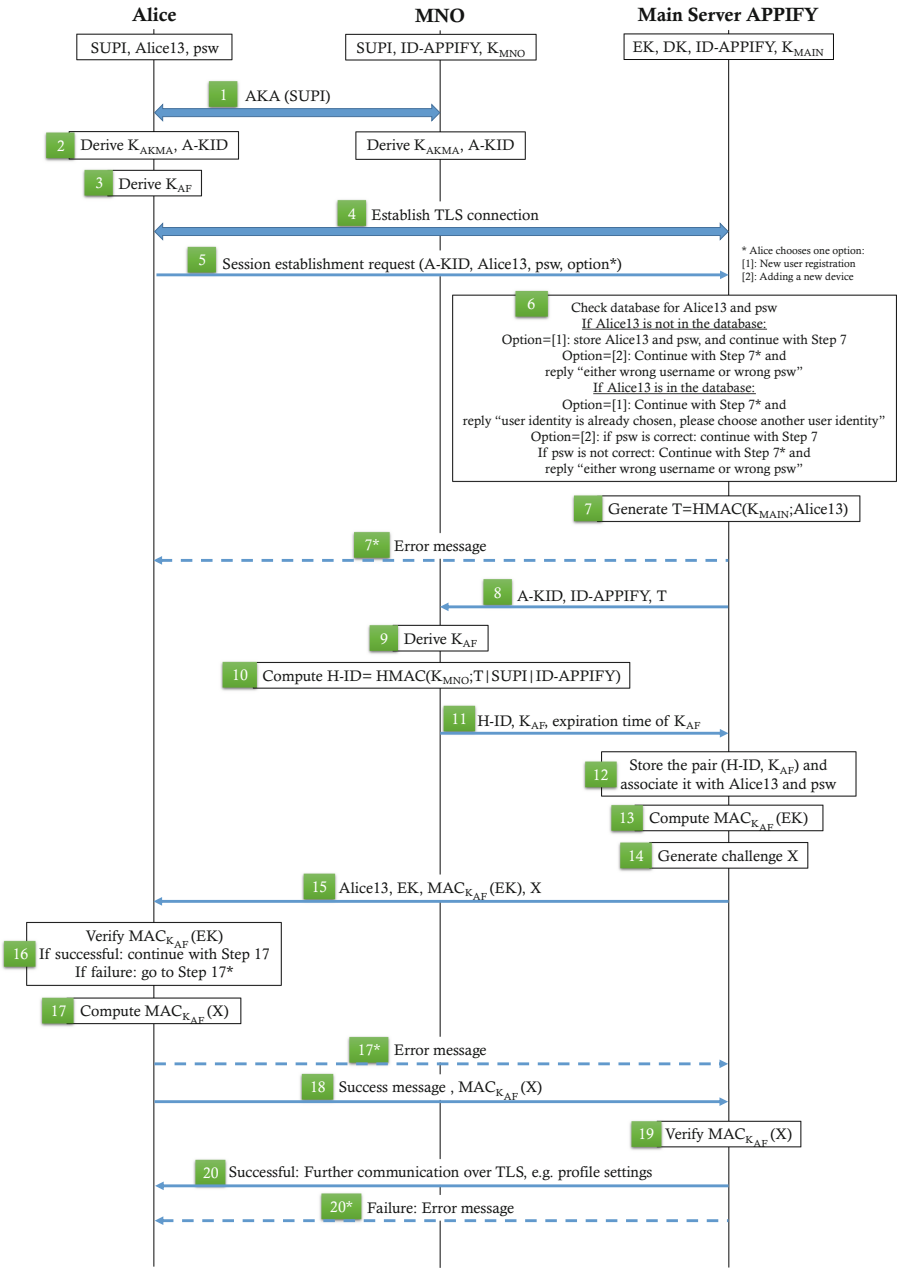

Fig. 3. Communication flow of Protocol A

1. Alice runs Authentication and Key Agreement (AKA) with MNO. (If an existing and valid shared key K_{AUSF} is stored, this step can be skipped.)
2. After AKA is completed, Alice and MNO (Authentication Server Function (AUSF) in MNO) compute K_{AKMA} from the shared key K_{AUSF}, which is agreed in AKA, and generate the AKMA Key Identifier (A-KID) [1].
3. Alice derives K_{AF} for APPIFY, according to AKMA specifications.
4. Alice establishes a TLS connection to S-APPIFY.
5. Alice sends session establishment request with the parameters A-KID, Alice13, psw, and the selected option to S-APPIFY through the secure channel. The options are (1) New user registration and (2) Registering a new device. Recall that a new device means a new SUPI.
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 to 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 generates T=$HMAC_{K_{MAIN}}$(Alice13).

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

8. S-APPIFY sends A-KID, ID-APPIFY, and T to MNO.
9. MNO derives K_{AF} according to AKMA specifications.
10. MNO also computes H-ID=$HMAC_{K_{MNO}}(T \,|\, SUPI \,|$ ID-APPIFY).
11. MNO sends H-ID, K_{AF}, and the expiration time of K_{AF} to S-APPIFY.
12. S-APPIFY stores the pair (H-ID, K_{AF}) and associates it with Alice13 and psw. It should be noted that MNO can enable the authentication of Alice to S-APPIFY without learning the identifier Alice13.
13. S-APPIFY computes $MAC_{K_{AF}}(EK)$.
14. S-APPIFY generates a challenge X to later verify that Alice has the shared key.
15. S-APPIFY sends Alice13, EK, $MAC_{K_{AF}}(EK)$, and X to Alice over the TLS channel.
16. Alice verifies $MAC_{K_{AF}}(EK)$.
    If the verification is successful: The protocol continues with Step 17.
    If the verification is a failure: The protocol continues with Step 17*.
17. Alice computes $MAC_{K_{AF}}(X)$.

17*. Alice sends an error message to S-APPIFY, and the protocol is aborted.

18. Alice sends $MAC_{K_{AF}}(X)$ to S-APPIFY along with the success message.

19. S-APPIFY verifies $MAC_{K_{AF}}(X)$.
    If the verification is successful: The protocol continues with Step 20.
    If the verification is a failure: The protocol continues with Step 20*.
20. Further communication continues over TLS; for example, the profile settings can be updated.

20*. S-APPIFY sends an error message to Alice and the protocol is aborted.

4.2 Protocol B: Signing In

Protocol B is run when the user and its 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 B runs between Alice and S-APPIFY and includes MNO. The protocol is explained step by step below, and the communication flow is shown in Fig. 4.

1. Alice runs AKA with MNO. (If an existing and valid K_{AUSF} is stored, this step can be skipped.)
2. Alice and MNO compute K_{AKMA} and generate the AKMA Key Identifier (A-KID).
3. Alice derives K_{AF} for APPIFY.
4. Alice establishes a TLS connection to S-APPIFY.
5. Alice sends session establishment request with A-KID, 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. S-APPIFY checks the database for Alice13.
   If Alice13 is found in the database: The protocol continues with Step 7,
   If Alice13 is not found in the database: The protocol continues with Steps 14–17 and then goes to Step 19*.
7. S-APPIFY finds 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.
8. S-APPIFY generates T=$HMAC_{K_{MAIN}}$(Alice13).
9. S-APPIFY sends A-KID, ID-APPIFY, and T to MNO.
10. MNO derives K_{AF}.
11. MNO also computes H-ID=$HMAC_{K_{MNO}}(T \mid SUPI \mid \text{ID-APPIFY})$.
12. MNO sends H-ID, K_{AF}, and the expiration time of K_{AF} to S-APPIFY.
13. S-APPIFY checks if the H-ID sent by MNO matches any existing H-ID(s) found in Step 7.
    If H-ID matches: The protocol continues with Step 14,
        If there is an existing K_{AF}, update it.
        If there is no existing K_{AF}, save it.
    If H-ID does not match: The protocol continues with Steps 14–17 and then goes to Step 19*.
14. S-APPIFY generates a challenge X.
15. S-APPIFY sends X to Alice over the TLS channel.

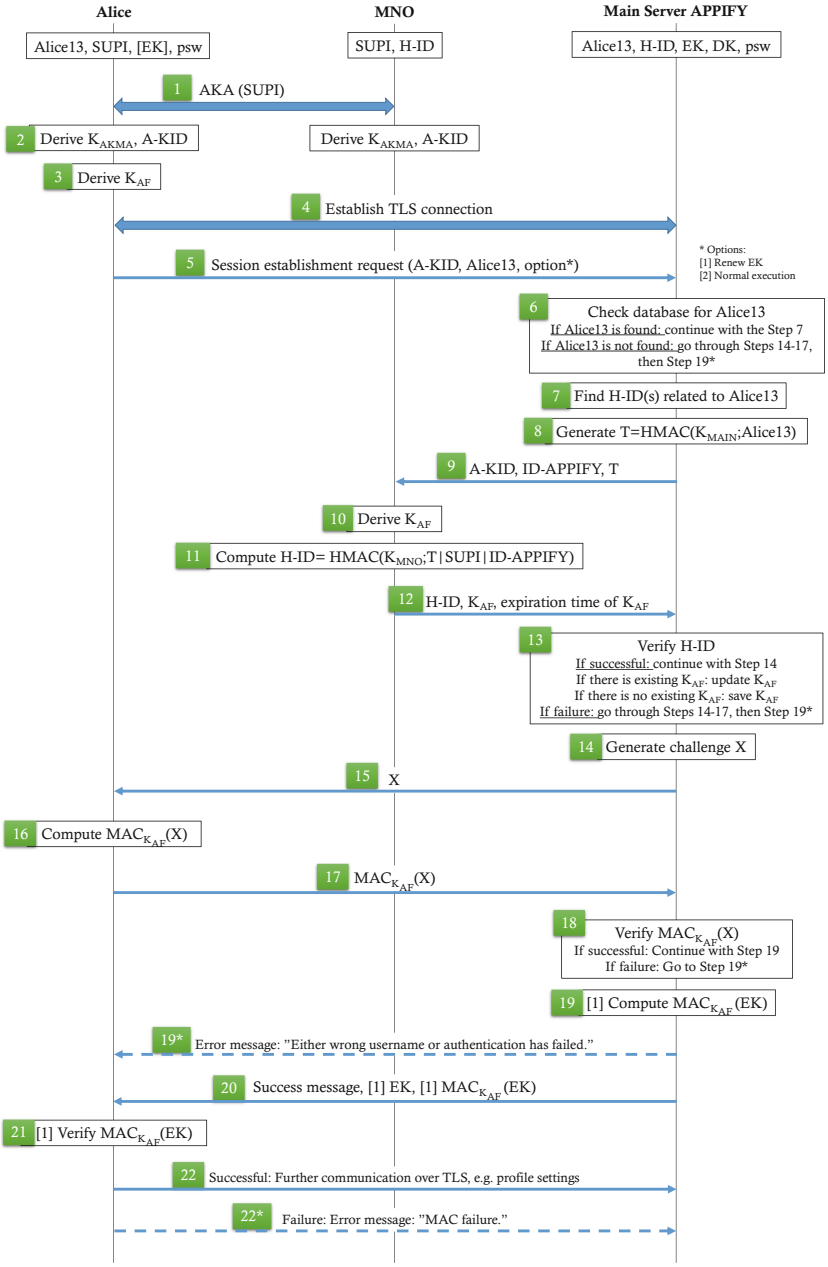

Fig. 4. Communication flow of Protocol B

16. Alice computes $MAC_{K_{AF}}(X)$.
17. Alice sends $MAC_{K_{AF}}(X)$ to S-APPIFY.
18. S-APPIFY verifies $MAC_{K_{AF}}(X)$.
    If the verification is successful: The protocol continues with Step 19.
    If the verification is a failure: The protocol continues with Step 19*.
19. If Option (1) is chosen, S-APPIFY computes $MAC_{K_{AF}}(EK)$. Otherwise, skip this step.

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

20. S-APPIFY sends the success message to Alice. If Option (1) is chosen, S-APPIFY also sends EK and $MAC_{K_{AF}}(EK)$.
21. If Option (1) is chosen, Alice verifies $MAC_{K_{AF}}(EK)$.
    If the verification is successful: The protocol continues with Step 22.
    If the verification is a failure: The protocol continues with Step 22*.
    If Option (2) is chosen, the protocol continues with Step 22.
22. Further communication continues over TLS; for example, the profile settings can be updated.

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

4.3 Protocol C: Connecting to Application in MEC Host

Protocol C is run when 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 C is explained step by step below, and the communication flow is shown in Fig. 5.

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 public key EK and saved Alice13, psw, H-ID, and K_{AF} in the database. MEC and M-APPIFY do not have any information on the user.

1. Alice and MEC establish a secure connection, e.g., DTLS connection.
2. Alice generates a random X. Then, she encrypts X and her identifier Alice13 by using the public key of S-APPIFY EK: $E_{EK}(Alice13, X)$.
3. Alice sends the ID-APPIFY and $E_{EK}(Alice13, X)$ to MEC through the secure channel.
4. MEC notices the identifier of S-APPIFY and forwards the message $E_{EK}(Alice13, X)$ to M-APPIFY.
5. M-APPIFY forwards $E_{EK}(Alice13, X)$ to S-APPIFY.
6. S-APPIFY decrypts $E_{EK}(Alice13, X)$ with its secret key DK. Then, S-APPIFY checks the database for Alice13.
   If Alice13 is in the database: S-APPIFY gets the related identifiers and keys, e.g., K_{AF}.
   If there is existing K_{AF}, the protocol continues with Step 7.

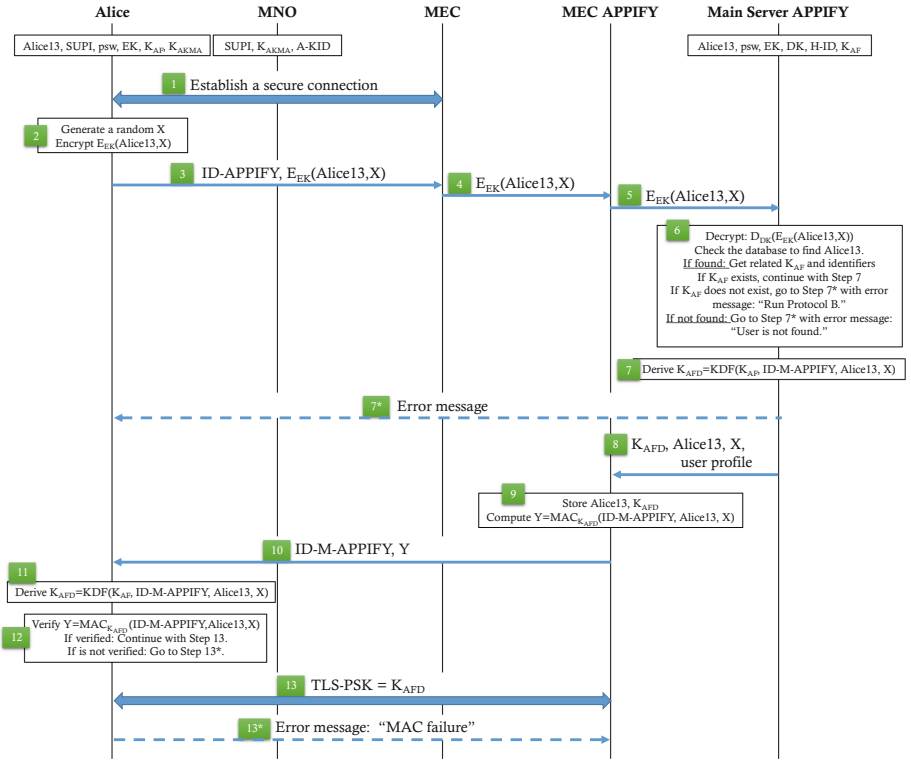

Fig. 5. Communication flow of Protocol C

If there is no existing K_{AF}, the protocol continues with Step 7* with the error message: "Run Protocol B."

If Alice13 is not in the database: The protocol continues with Step 7* with the error message: "User is not found."

7. S-APPIFY derives a new key K_{AFD}=KDF(K_{AF}, ID-M-APPIFY, Alice13, X), where the ID-M-APPIFY is the identifier of M-APPIFY.

7* The error message is sent and the protocol is aborted.

8. S-APPIFY sends K_{AFD}, Alice13, X, and the user profile of Alice13 to M-APPIFY.
9. M-APPIFY stores Alice13 and K_{AFD}. Then, it computes Y=$MAC_{K_{AFD}}$(ID-M-APPIFY, Alice13, X).
10. M-APPIFY sends ID-M-APPIFY and Y to Alice through MEC.
11. Alice derives K_{AFD}=KDF(K_{AF}, ID-M-APPIFY, Alice13, X) after receiving ID-M-APPIFY.
12. Alice computes $MAC_{K_{AFD}}$(ID-M-APPIFY, Alice13, X) and verifies whether it matches Y.

If the verification is successful: The protocol continues with Step 13.

If the verification is a failure: The protocol continues with Step 13*.

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

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

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 use the same M-APPIFY after some time, assuming that the time limit is not over. In this case, the following steps are followed.

Is the TLS connection still on?

Yes: Protocol C continues with Step 13.

No: Is K_{AFD} is still valid?

Yes: TLS session resumption [18] is run with the key K_{AFD}, and Protocol C continues with Step 13.

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

4.4 Protocol D: Changing the MEC Host

Protocol D is run when Alice, who is actively using M-APPIFY, i.e., the parties are in Step 13 of Protocol C, 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. This paper focuses on the case where MNO does not change and leaves the extension to the roaming case for future work.

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 B 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 [8]. Protocol D is explained step by step below, and the communication flow is shown in Fig. 6.

0. Protocol starts when there is an existing connection between Alice and source M-APPIFY resulting from Protocol C, i.e., Step 13.
1. When MEO determines the target MEC host, MEO provides this information to the source MEC host. Then, 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 is the same state information that was mentioned in the Introduction, in Sect. 1.

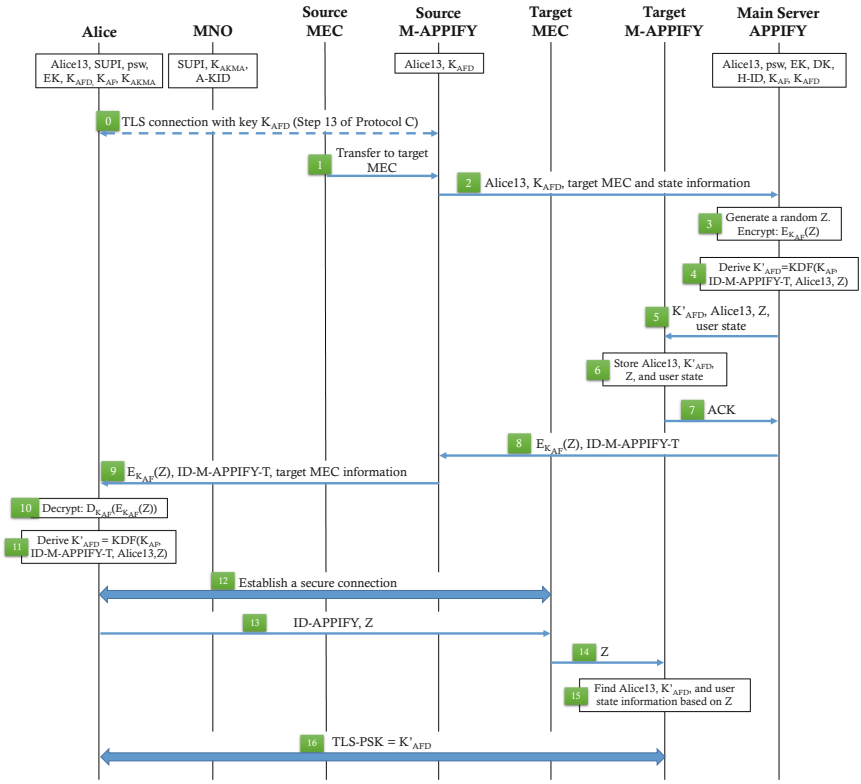

Fig. 6. Communication flow of Protocol D

3. S-APPIFY generates a random Z. Then, it encrypts Z with K_{AF}: $E_{K_{AF}}(Z)$.
4. S-APPIFY derives the key K'_{AFD}=KDF(K_{AF}, ID-M-APPIFY-T, Alice13, Z), where ID-M-APPIFY-T is the identifier of the target M-APPIFY.
5. S-APPIFY sends K'_{AFD}, Alice13, Z, user profile, and service state information of Alice13 to the target M-APPIFY.
6. Target M-APPIFY stores Alice13, K'_{AFD}, Z, user profile, and service state information of Alice13.
7. Target M-APPIFY sends an acknowledgment to S-APPIFY that it received the message.
8. S-APPIFY sends $E_{K_{AF}}(Z)$ and ID-M-APPIFY-T to the source M-APPIFY.
9. Source M-APPIFY sends $E_{K_{AF}}(Z)$, ID-M-APPIFY-T, and the target MEC host information to Alice.
10. Alice decrypts $E_{K_{AF}}(Z)$ with K_{AF} to retrieve Z.
11. Alice derives K'_{AFD}=KDF(K_{AF}, ID-M-APPIFY-T, Alice13, Z).
12. Alice establishes a secure connection with the target MEC host, e.g., DTLS connection.
13. Alice sends Z with ID-APPIFY to the target MEC host.
14. Target MEC host forwards Z to the target M-APPIFY.

15. Target M-APPIFY checks the database for Z and finds related Alice13, K'_{AFD}, and the user information.
16. Target M-APPIFY initiates a TLS-PSK connection with Alice with the key K'_{AFD}.

When the TLS connection between Alice and target M-APPIFY is established successfully, target M-APPIFY continues with providing the services to Alice instead of 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.

5 Analysis

Our solution uses 3GPP AKMA for the authentication and key sharing between the user Alice and the main server of the application S-APPIFY in Protocol A (Sect. 4.1) and Protocol B (Sect. 4.2). The shared key resulting from AKMA is used later in Protocol C (Sect. 4.3) and Protocol D (Sect. 4.4) when Alice wants to get service from the M-APPIFY. It is appropriate to adapt the AKMA system to our solution 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.

Moreover, AKMA is a better option for the 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 synchronization of the keys between these parties requires less effort with AKMA.

It is natural to use usernames and passwords for accessing the applications. However, when Alice uses AKMA, she does not need to enter her password for the application all the time she uses the application. This 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 A and B, a TLS connection is established between Alice and the main server of APPIFY before she sends her identifiers and password. This way, Alice can be sure that she is talking to the correct server. Moreover, these identifiers and passwords would not be visible to outsiders, and neither to MNO.

Protocols C and D focus on the usage of APPIFY in MEC hosts. Therefore, there should also be a secure connection between Alice and MEC, e.g., DTLS.

The handshake can be based on a pre-shared key (resulting from AKMA) or a certificate. 3GPP defines usage of AKMA in their technical documents [3]. We do not discuss how to secure the connection between the user and MEC further.

3GPP studies focus on the authentication between Alice and MEC host and Alice and MEC system. In this paper, we focus on authentication at the application level. We use AKMA to provide mutual authentication between the user and the main server of the application. As a product of AKMA, the user and the main server of the application share a key. Another 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 C and D. This newly derived key depends on the MEC application. Therefore, every time 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 the need to run AKMA each time. In addition, since the same key is not used with different entities, the key is less likely to be compromised.

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 would like to learn as much as possible without interfering with the communication protocol [15]. In order to preserve the privacy of the user, MNO and MEC should not learn anything related to the user that is not needed for providing services.

Privacy of the user is preserved against MNO with our solution. Therefore, MNO only learns that Alice uses S-APPIFY since 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, and her password, because these are sent to S-APPIFY in Protocols A and B after the TLS connection is established. Similarly, in Protocols C and D, Alice first establishes a secure connection with MEC and then sends her identifier, which is also encrypted with the public encryption key of the main server. This latter encryption protects the privacy of the identifier against honest-but-curious MEC hosts.

We have introduced H-ID to AKMA to prevent MNO from learning the identifier Alice13 during the authentication. S-APPIFY sends the keyed hash of Alice13 instead of Alice13 itself to MNO, and MNO binds A-KID and SUPI to the keyed hash of Alice13. Therefore, neither MNO nor S-APPIFY can link SUPI and Alice13.

In addition, the communication between Alice and S-APPIFY continues over the TLS channel in Protocols A and B, so MNO cannot see the data sent between the two parties. In Protocols C and D, 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 C and D, we introduce 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 the communication of Alice with either S-APPIFY or M-APPIFY.

Privacy of the user is also preserved against MEC. Alice sends her identifier Alice13 after encrypting it with the public encryption key of the main server. Also, 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 the identifiers of Alice or the details of the services she gets from M-APPIFY.

6 Conclusion and Future Work

Multi-access edge computing (MEC) is an emerging technology in 5G. We solve the problem of creating a secure and privacy-preserving communication channel between the user and MEC application in the local MEC host when the user moves from the source MEC host to another target MEC host. The scope of the paper is restricted to the case where the source and the target MEC hosts belong to the same MNO.

One limitation of our study is that AKMA is not yet deployed. Therefore, observing the limits of AKMA in practice and experimenting with our solution are left for future work. Two other topics for future work are the formal verification of our solution and extending the scope to the roaming situation where the source and the target MEC hosts belong to different MNOs.

Acknowledgements. The authors would like to thank Mohsin Khan for his helpful comments.

References

1. 3GPP. Authentication and key management for applications (AKMA) based on 3GPP credentials in the 5G system (5GS). Technical Specification TS 33.535 V17.4.0, 3GPP (2021)
2. 3GPP. Security aspects of enhancement of support for enabling edge applications. Technical Specification TS 33.558 V0.3.0, 3GPP (2021)
3. 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 (2021)
4. Ali, A., Lin, Y.D., Li, C.Y., Lai, Y.C.: Transparent 3rd-party authentication with application mobility for 5G mobile edge computing. In: 2020 European Conference on Networks and Communications (EuCNC), pp. 219–224. IEEE, Dubrovnik (2020). https://doi.org/10.1109/EuCNC48522.2020.9200937
5. 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 **10**, 35106–35119 (2022). https://doi.org/10.1109/ACCESS.2022.3162851
6. Ali, B., Gregory, M.A., Li, S.: Multi-access edge computing architecture, data security and privacy: a eview. IEEE Access **9**, 18706–18721 (2021). https://doi.org/10.1109/ACCESS.2021.3053233

7. ETSI. Application Mobility Service API. Group Specification GS MEC 021 V2.2.1, ETSI (2022)
8. ETSI. Framework and Reference Architecture. Group Specification GS MEC 003 V3.1.1, ETSI (2022)
9. Huang, X., Tsiatsis, V., Palanigounder, A., Su, L., Yang, B.: 5G authentication and key management for applications. IEEE Commun. Stand. Magaz. **5**(2), 142–148 (2021). https://doi.org/10.1109/MCOMSTD.001.2000024
10. Lei, W., et al.: 5G security system design for all ages. In: 5G System Design, pp. 341–390. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-73703-0_6
11. 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. **7**(8), 6722–6747 (2020). https://doi.org/10.1109/JIOT.2020.3004500
12. Liyanage, M., Porambage, P., Ding, A.Y., Kalla, A.: Driving forces for multi-access edge computing (MEC) IoT integration in 5G. ICT Express **7**(2), 127–137 (2021). https://doi.org/10.1016/j.icte.2021.05.007
13. Nencioni, G., Garroppo, R.G., Olimid, R.F.: 5G multi-access edge computing: security, dependability, and performance. arXiv:2107.13374 [cs] (2021)
14. Niewolski, W., Nowak, T.W., Sepczuk, M., Kotulski, Z.: Token-based authentication framework for 5G MEC mobile networks. Electronics **10**(14) (2021). https://doi.org/10.3390/electronics10141724
15. Paverd, A., Martin, A., Brown, I.: Modelling and Automatically Analyzing Privacy Properties for Honest-But-Curious Adversaries. Technical Report (2014)
16. Ranaweera, P., Jurcut, A., Liyanage, M.: MEC-enabled 5G use cases: a survey on security vulnerabilities and countermeasures. ACM Comput. Surv. **54**(9), 1–37 (2022). https://doi.org/10.1145/3474552
17. Ranaweera, P., Jurcut, A.D., Liyanage, M.: Survey on multi-access edge computing security and privacy. IEEE Commun. Surv. Tutor. **23**(2), 1078–1124 (2021). https://doi.org/10.1109/COMST.2021.3062546
18. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. Internet-Draft draft-ietf-tls-rfc8446bis-03, IETF (2021, work in progress)
19. Sabella, D.: MEC federation and mobility aspects. In: Multi-access edge computing: software development at the network edge. TTE, pp. 245–279. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79618-1_8
20. Sanchez-Gomez, J., Marin-Perez, R., Sanchez-Iborra, R., Zamora, M.A.: MEC-based architecture for interoperable and trustworthy internet of moving things (2022). https://doi.org/10.1016/j.dcan.2022.03.028
21. Tabatabaee Malazi, H., et al.: Dynamic service placement in multi-access edge computing: a systematic literature review. IEEE Access **10**, 32639–32688 (2022). https://doi.org/10.1109/ACCESS.2022.3160738