Prototype Implementation of a 5G Group-Based Authentication and Key Agreement Protocol

Markus Ahlström
ims11mah@student.lu.se
Simon Holmberg
simon.holmberg632@gmail.com

Department of Electrical and Information Technology
Lund University

Advisors:
Christian Gehrmann, SICS Swedish ICT/Lund University
Rosario Giustolisi, SICS Swedish ICT

December 12, 2016

Printed in Sweden
E-huset, Lund, 2016

Abstract

The number of machine-type communications (MTC) devices is predicted to increase enormously in the near future, and this increase is expected to be accommodated by 5G. However, the Authentication and Key Agreement (AKA) mechanism used in the fourth generation of mobile telecommunications systems, EPS, may prove too inefficient for handling a large number of MTC devices simultaneously trying to connect. In order to address this issue a number of group-based authenticated key agreement protocols have been proposed. In this thesis we review six of the proposed protocols. For the purpose of implementing such a protocol in a realistic setting, we also review eight open-source EPS platforms of which one was chosen as a basis for an implementation.

The main contribution of this thesis is a prototype implementation of a group-based authenticated key agreement protocol designed by Giustolisi et al. Moreover, we provide a detailed specification that aims to serve as a reference for a possible standardization of the new protocol. The secondary contribution of this thesis is a performance analysis of the protocol.

Acknowledgements

We would like to thank our friends and families for their encouragement during this degree project. We also thank our supervisors Rosario Giustolisi and Christian Gehrmann for their guidance and support. Lastly, we thank everyone at SICS Swedish ICT for providing us with a stimulating work place.

Table of Contents

Nomenclature

3GPP	3rd Generation Partnership Project
AAA	Authentication, Authorization and Accounting
AAA	Authentication, Authorization and Accounting
AIA	Authentication Information Answer
AIR	Authentication information request
AK	Anonymity key
AKA	Authentication and Key Agreement
AMF	Authentication Management Field
AS	Access Stratum
AuC	Authentication Centre
AUTN	Authentication Token
AV	Authentication Vector
AVP	Attribute Value Pair
CK	Ciphering Key in 3G
CN	Core Network
EMM	EPS Mobility Management protocol
eNodeB	E-UTRAN Node B
EPS	Evolved Packet System
ESM	EPS Session Management protocol
GUTI	Globally Unique Temporary Identity
HSS	Home Subscriber Server

IE	Information Element
IK	Integrity Key in 3G
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
K_{ASME}	Local master key in EPS
LTE	Long Term Evolution
MAC	Message Authentication Code
MCC	Mobile Country Code
ME	Mobile Equipment
MME	Mobility Management Entity
MNC	Mobile Network Code
MSIN	Mobile Subscriber Identification Number
MTC	Machine-Type Communications
NAS	Non-Access Stratum
OAI	OpenAirInterface
PLMN-ID	Public Land Mobile Network Identity
RAND	Random 128-bit string
SN_{ID}	Serving Network identity
SQN	Sequence number used in AKA
UE	User Equipment
UICC	Universal Integrated Circuit Card
UMTS	Universal Mobile Telecommunications System
USIM	Universal Subscriber Identity Module
XRES	Expected Response

List of Figures

List of Tables

Chapter 1

Introduction

The 5th generation of mobile telecommunications systems (5G) is under development and envisioned to provide higher bandwidth and lower latency than its predecessors, i.e. the third-generation Universal Mobile Telecommunications System (UMTS) and the fourth-generation Evolved Packet System (EPS) [3]. So far, mobile telecommunication has been dominated by mobile telephony and the provision of Internet connectivity to user-controlled terminals. The 3rd generation paved the way for smartphones and the 4th generation further increased data rates.

With the fifth generation, the aim is to incorporate a broader part of society such as manufacturing, smart grids and e-health into the network. A lot of the future applications envisioned concerns connected devices that have no direct user interaction, called machine-type communications (MTC) devices [3][4][5]. As the number of connected devices in 5G consequentially will not be limited by the number of people, the number of connected devices might increase enormously. However, this is a huge technical challenge not only because the devices are potentially so numerous, but also because devices not attended by humans cannot be expected to behave in ways that the human-attended devices have been behaving.

For example, a large number of MTC devices can be programmed to synchronously attach to the network. The challenge that this scenario constitutes has been identified and studied. Because the devices attach synchronously, the attachment signalling might lead to congestion in the radio access network, congestion in the core network and overload of authentication servers. According to Ericsson [6], there have already been cases where millions of MTC synchronised attachments have been made in mobile networks, causing authentication avalanches which produced peaks of up to 40 times more signalling in the network than expected during the busy hour.

When a device attaches to a mobile telecommunications network, it is required that the device and the network be mutually authenticated and that they agree on a session key so that integrity and confidentiality of data can be obtained. The protocol used for this in the 4th generation, EPS Authentication and Key Agreement (AKA), accounts for a large part of each attachment procedure, including signalling in the backhaul of the network. Optimizing this protocol for MTC devices could prove to be a solution to the aforementioned challenge.

One class of optimizations is group-based AKA protocols, which are protocols that let a group of MTC devices authenticate to the mobile network based on shared attributes. Several group-based AKA protocols have been proposed in research articles in recent years [7][8][9][10]. However, none of them has emerged as an obvious future standard solution, so the field is still open to new suggestions. Two criteria that a new solution must meet are that it is practical and that the gain in performance when applied to the new MTC scenario is sufficient. Thus, it is important that new solutions be evaluated with regards to these factors.

This degree project was done at the security lab of the research institute SICS Swedish ICT. The lab participates in the security project 5G-ENSURE, which is a part of the 5G Infrastructure Public Private Partnership (5G-PPP). Within this project, the lab is doing research on potential security enablers for 5G in the area of authentication, authorization and accounting (AAA). One such enabler that the lab is developing is a group-based AKA protocol.

1.1 Objectives

The primary goal of this degree project is to produce a prototype implementation of an early version of the aforementioned group-based AKA protocol. The first part of the project is to review group-based AKA protocols proposed in related work. The second part is to find and evaluate open-source mobile telecommunications platforms on which the protocol can be implemented in a convincing manner. The third part is to implement the protocol. One goal of this part is to be able to produce a specification in support of the protocol. The final part is to analyse how well the protocol performs regarding latency and data traffic.

1.2 Limitations

It is not within the scope of this thesis to analyse the security of the protocol.

This thesis considers only *open-source* mobile telecommunications platforms for the task of implementing the group-based AKA protocol. In other words, no proprietary platforms, even though they might otherwise be good choices, are considered.

This thesis considers the implementation of a *successful* group-based authentication procedure. Any divergence from the straight-forward cases is out of scope, including handling of unknown, unforeseen or erroneous protocol data. For example, we do not consider how the protocol relates to handover procedures.

For our purposes, a protocol run is completed when the user equipment (UE) and the mobility management entity (MME) has authenticated each other and produced the local master key. Any protocol behaviour exceeding this phase, such as the derivations of further keys, is out of scope.

Furthermore, all references to the 3GPP standardization of the EPS are to the Release 10 version of the standard.

Finally, the storage of new parameters that would reside in the UICC is not considered in detail, because the EPS platform used in this thesis abstracted away the UICC file system.

Chapter 2

Background

This chapter is dedicated to introducing the underlying context on which the thesis is built. It is centred around the Authentication and Key Agreement (AKA) protocol in modern telecommunication networks; therefore, this chapter introduces how the AKA procedure is currently performed in such a network. This includes presenting terms, main entities and mechanics used by the procedure. The chapter provides a background in order to understand the related work and implementation chapters in the thesis.

2.1 Evolved packet system – introduction, terms and entities

Evolved Packet System (EPS) is the name for a fourth-generation mobile telecommunications network architecture that is specified in [11]. The architecture usually involves a mobile device using a radio interface in order to communicate with a gateway node, which supplies it with Internet connectivity. It is sometimes known by the brand name Long Term Evolution (LTE), which in more technical contexts (including this thesis) refers only to the new radio access technology standard used in the EPS.

The EPS is an evolution of 3G UMTS, the previous generation of mobile telecommunications networks. It offers higher data rates and lower latencies than its predecessor and supports multiple radio access technologies [12]. It consists of multiple entities running a set of protocols in order to supply users with mobile network connectivity and services.

The security approach of EPS is based on the previous generation of mobile telecommunication networks and has inherited the requirements of authentication and privacy. The security has evolved with the help of public review of security functions. The result of this is a standardization of security procedures that has had great success. The organization that standardizes the EPS is called the 3rd Generation Partnership Project (3GPP). The standardization is a mix of clearly defined protocols and abstract procedures. Some aspects are standardized to ensure interoperability, while others are not standardized, which allows for differentiation and quick changes in order to extend to new technologies.

The EPS AKA protocol is the AKA protocol used in the EPS [13]. In order to present the EPS AKA, we first introduce the entities which are involved.

2.1.1 User equipment

A user equipment (UE) is an entity which enables network access to a user by a radio interface [12]. It is defined as the combination of a mobile equipment (ME) and a Universal Integrated Circuit Card (UICC). The ME includes the terminal which can run access protocols, and the UICC is a physically secure smart card platform on which a Universal Subscriber Identity Module (USIM) application can be run. The USIM is an application used to register to certain network services using appropriate security. A typical example of a UE is a mobile phone including a USIM card.

IMSI, IMEI and GUTI

The UE contains various identities which are defined in clause 2 of TS 23.003 [14]. Firstly, it stores the international mobile subscriber identity (IMSI), which is an identity parameter stored in the UICC. As the name suggests the IMSI uniquely identifies a mobile subscriber of the mobile communications network. It consists of three fields: the mobile country code (MCC), the mobile network code (MNC) and the mobile subscriber identification number (MSIN). The MCC and MNC internationally identifies the mobile network operator of the subscriber and the MSIN identifies the subscriber. The international mobile equipment identity (IMEI) is an identity which uniquely identifies the ME. Lastly, a globally unique temporary identity (GUTI) is a temporary identity whose purpose is to unambiguously identify a subscriber in the network.

MTC device

A machine-type communications (MTC) device can be defined as a terminal that is made for MTC, which means that it is not attended by humans. In most of this thesis and in at least one 3GPP specification document [15], it is assumed to be a UE, which means that it includes a UICC.

2.1.2 Serving network

The serving network is the network providing the immediate network access to the UE. Which serving network that a UE is connected to depends on where the UE is located at the moment.

2.1.3 E-UTRAN Node B

The Evolved Universal Terrestrial Radio Access Network Node B (E-UTRAN Node B, or eNodeB) is the base station that communicates directly with the UE over radio and provides it with connectivity to the backhaul of the EPS. [16].

2.1.4 Mobility management entity

The mobility management entity (MME) is an entity on the serving network that manages mobility, sessions and authentication of UEs.

Public Land Mobile Network Identity

The Public Land Mobile Network Identity (PLMN-ID) identifies a Public Land Mobile Network Operator, which is the operator providing services over an air interface [12]. In this thesis, we also refer to the PLMN-ID as SN_{ID} when the PLMN is a serving network.

2.1.5 Home Subscriber Server

The Home Subscriber Server (HSS) is the entity that stores all subscription data of the subscribers to an operator [17]. It resides in the home network of the subscriber. The home network, in contrast with the serving network, is the network that is controlled by the subscriber's operator. The HSS includes storage of security credentials and identification parameters, typically in an authentication centre (AuC). The HSS is also responsible for receiving and processing authentication requests of UEs sent by MMEs. It identifies UEs and generates authentication data which then is sent to the MME in question.

2.1.6 User plane and control plane

The network is usually divided into two conceptual planes. One is called the user plane and involves all transport of user data, e.g. speech, in the network. The other is called the control plane (sometimes called the signalling plane) and includes all protocols which transfer control data, e.g. attachment and detachment of UEs.

2.1.7 Access and core network, Access Stratum (AS) and Non-Access Stratum (NAS)

The access network provides the UE with immediate connectivity (through wireless or wire). The core network (CN) is the part of the telecommunications network that is not part of the access network. It includes both the serving network and the home network, both the MME and the HSS [18]. Furthermore, the communication between the UE and the network is conceptually divided into two strata: the Access Stratum (AS), which includes protocols for communication between the UE and the access network; and the Non-Acccess Stratum (NAS), which includes protocols for communication between the UE and the core network.

2.1.8 Trust model

Based on [19] we illustrate a trust model of EPS in Figure 2.1. It can be summarized as illustrating that the CN is in a trusted environment and the AS is not. In other words, the CN is assumed to run protocols such as Internet Key Exchange and the

IP security protocol which ensures that the communication of these elements are secure [13]. The untrusted part of the model is the AS, radio access network, which is the communication between the UE and the eNodeB using a radio interface. This communication can be intercepted and modified by an attacker and is therefore untrusted. The main reason behind an authentication an key agreement protocol being deployed here is in order to secure the AS and provide mutual authentication between MME and UE.

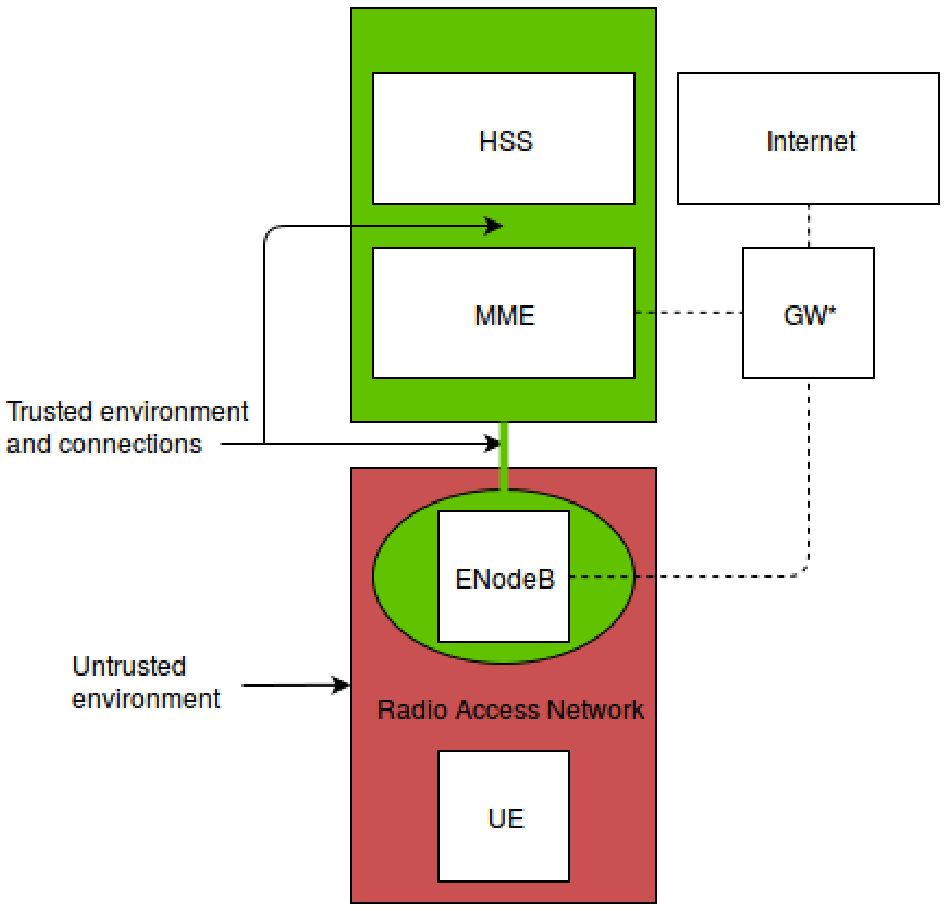


Figure 2.1: Overview of the trust model of EPS.

2.1.9 AAA

Authentication, authorization and accounting (AAA) is a term used to describe a framework with a set of properties that are desirable in networks that need both connection security, policy security and a platform for business models. "Authentication" describes the need for two communicating nodes to ensure that the other node is authentic – not an impersonator. "Authorization" has to do with ensuring that certain nodes only have access to resources it is allowed to. Lastly, "accounting" is the property that when a certain node does something in the network it can later be proven, preferably so it can be convincing in a court of law.

2.2 EPS AKA

The EPS AKA protocol is an enhancement of UMTS AKA. It uses a AAA framework, credential infrastructures (e.g the USIM, MME and HSS) together with an authenticated key agreement mechanism in order to solve, among other things, the trust issue described in Section 2.1.

The EPS AKA procedure itself is described below and is based on its description in [13], [1] and [18].

2.2.1 Prerequisites

The security behind the EPS AKA protocol is based on a symmetric 128-bit key shared between the HSS and the UICC smart card, hereafter denoted by **K**. This key is uniquely associated to an IMSI. Furthermore, it is assumed that a SN_{ID} has already been communicated to the UE before the protocol is initiated.

2.2.2 Key derivation functions used in EPS

The EPS uses a key hierarchy based on the root key **K**. In order to derive father keys and security parameters, a set of functions which use **K** as input have been suggested by the 3GPP. The most central functions of that sort are named f1, f2, f3, f4 and f5 and are described in [20]. The output of these functions are used for both authentication and in order to derive the local master key K_{ASME}, which is done through another function called KDF. The K_{ASME} key is then used to derive a multitude of children keys for the purpose of NAS and AS ciphering and integrity protection. One motivation for generating different keys from one master key is that if one child key is compromised then the others would still be safe and another motivation is that key changes or updates are easier to perform without changing the master key.

An example algorithm set for the f1–f5 functions is provided by 3GPP in [2] and is called MILENAGE. The algorithms allow each operator to have a unique implementation of the functions by specifying a certain value in the algorithms which is called the Operator Variant Algorithm Configuration Field (OP).

2.2.3 EPS AKA procedure description

Assuming now that a UE, holding **K** and IMSI in its UICC, wants to connect to EPS services with the help of the MME, we describe a typical EPS AKA procedure as seen in Figure 2.2.

1. Attach request (UE –> MME)

The first message sent in EPS AKA is the Attach Request message wherein the UE requests access to EPS services from the serving network MME. The message includes UE supported capabilities and the IMSI (or GUTI or IMEI, but those cases we will not describe) of the UE. The UE in this stage cannot trust the serving network.

2. Authentication information request (MME –> HSS)

The MME receives the Attach Request. The MME in this stage cannot trust the UE, so it needs to authenticate that this UE is allowed to attach to its services. This is initiated by sending an Authentication Information Request (AIR) message to the HSS. This message includes the IMSI of the UE and an identifier for the serving network.

3. Generate UMTS authentication vector (HSS)

Upon receiving the AIR message, the HSS starts to prepare an authentication vector (AV) based on the shared secret **K** which is coupled with the IMSI.

The first authentication vector (AV) generated is called the UMTS AV, which is generated for legacy reasons. The vector consists of the expected authentication response (XRES), a ciphering key (CK), an integrity key (IK), a random challenge called RAND and an authentication token (AUTN). The AUTN consists of a parameter called the authentication management field (AMF), a message authentication code (MAC) and a sequence number (SQN) exclusive OR an anonymity key (AK). The MAC is made with **K**, AMF, SQN and RAND. The AK is constructed from **K** and RAND. The XRES is also constructed from **K** and RAND. See Table 2.1 for an overview of the UMTS AV parameters.

Table 2.1: UMTS AV parameters. Sizes of the parameters are based on MILENAGE [2].

Param.	**Produced by**	**Description**	**Size** (bits)
RAND	N/A	Authentication challenge	128
AMF	N/A	Unspecified, exists for future use	16
SQN	N/A	Synchronization value (UE/HSS)	48
XRES/RES	f2(**K**, RAND)	(Anticipated) response by the UE	64
CK	f3(**K**, RAND)	Used to generate K_{ASME}	128
IK	f4(**K**, RAND)	Used to generate K_{ASME}	128
MAC	f1(**K**, AMF, SQN, RAND)	Message authentication code	64
AK	f5(**K**, RAND)	Value used to conceal SQN	48
AUTN	(SQN exclusive or AK) || AMF || MAC)	Authentication token	128

4. Generate EPS authentication vector (HSS)

The UMTS AV is used to produce the EPS AV. A new parameter is generated by using a key derivation function with CK, IK, a serving network identifier (SN_{ID}) and SQN exclusive OR AK. The new parameter is a 256-bit local master key called

K_{ASME}. One improvement that is achieved by the introduction of K_{ASME} to the AKA is that SN_{ID} is bound to the encryption and integrity keys.

The EPS vector consists of AUTN, K_{ASME}, XRES and RAND. See Table 2.2 for an EPS AV parameter overview.

Table 2.2: EPS AV

Param.	**Produced by**	**Descr.**	**Size (bits)**
K_{ASME}	KDF(CK, IK, SQN XOR AK, SN_{ID})	Local master key	256
AUTN	See Table 2.1	See Table 2.1	See Table 2.1
XRES	See Table 2.1	See Table 2.1	See Table 2.1
RAND	See Table 2.1	See Table 2.1	See Table 2.1

5. Authentication information answer (HSS –> MME)

The HSS sends the EPS AV in the Authentication Information Answer (AIA) message to the MME.

6. Authentication request (MME –> UE)

The MME sends the message Authentication Request to the UE. The message includes the aforementioned AUTN and RAND. It also holds a key set identifier, which identifies the key set to be used between the MME and the UE.

7. Generate authentication response (UE)

Using the same set of algorithms as the HSS, the UE produces its own version of MAC, named XMAC. It compares these two values and if they are the same the UE has authenticated the serving network. The UE continues by producing a response parameter called RES, which is made in the same way as XRES. It also produces CK and IK and lastly K_{ASME} just as the HSS has done.

8. Authentication response (UE –> MME)

The UE sends the RES parameter to the MME in the Authentication Response message.

9. Mutual authentication complete (MME)

The MME checks that the received value of RES is the same as the XRES. If this is true then mutual authentication between the UE and the network has been achieved. A shared session local master key, K_{ASME}, has been established between the UE and the MME. This key can now be used to generate integrity keys and ciphering keys for both the NAS and the AS.

The EPS AKA message exchange and parameter generation can be seen in in Figure 2.2. Note that for the intents of this thesis the exchange is finished when

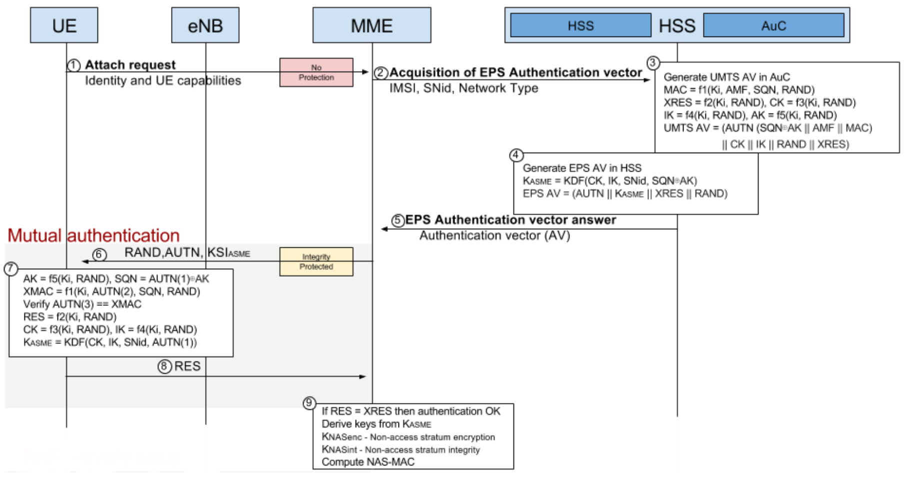

UE
eNB
MME
HSS
HSS
AuC
① Attach request
Identity and UE capabilities
No Protection
② Acquisition of EPS Authentication vector
IMSI, SNid, Network Type
③ Generate UMTS AV in AuC
MAC = f1(Ki, AMF, SQN, RAND)
XRES = f2(Ki, RAND), CK = f3(Ki, RAND)
IK = f4(Ki, RAND), AK = f5(Ki, RAND)
UMTS AV = (AUTN (SQN⊕AK || AMF || MAC) || CK || IK || RAND || XRES)
④ Generate EPS AV in HSS
KASME = KDF(CK, IK, SNid, SQN⊕AK)
EPS AV = (AUTN || KASME || XRES || RAND)
⑤ EPS Authentication vector answer
Authentication vector (AV)
Mutual authentication
⑥ RAND,AUTN, KSIASME
Integrity Protected
⑦ AK = f5(Ki, RAND), SQN = AUTN(1)⊕AK
XMAC = f1(Ki, AUTN(2), SQN, RAND)
Verify AUTN(3) == XMAC
RES = f2(Ki, RAND)
CK = f3(Ki, RAND), IK = f4(Ki, RAND)
KASME = KDF(CK, IK, SNid, AUTN(1))
⑧ RES
⑨ If RES = XRES then authentication OK
Derive keys from KASME
KNASenc - Non-access stratum encryption
KNASint - Non-access stratum integrity
Compute NAS-MAC

Figure 2.2: EPS AKA protocol up to mutual authentication (UE/CN) [1]

mutual authentication and master key agreement between the CN and the UE has been achieved. The protocol continues after this point by deriving integrity and encryption keys from the K_{ASME} used for NAS and AS signalling.

2.3 3GPP definitions

3GPP have defined certain elements and procedures that are considered the standard of EPS. We present a number of such 3GPP terms and definitions here in order to supply the reader with an understanding of the implementation presented in Chapter 6. In particular, the NAS protocols, the S6a interface and the USIM characteristics are presented.

2.3.1 Non-access stratum

Control plane messages between the UE and the MME are included in the NAS protocols, specifically the EPS mobility management (EMM) protocol and the EPS session management (ESM) protocol. These are also defined as layer 3 (L3) protocols. Their structure is described in the 3GPP documents [21] and [22]. Each message contains a number of parameters called information elements (IEs).

Information elements

IEs use a standard type-length-value (TLV) format and come in four different standardized types: TLV, LV, V and TLV-E [21]. "TLV-E" types simply uses a larger length value.

Table 2.3: Information element format

8	7	6	5	4	3	2	1	
Information element identifier								octet 1
Length of IE contents								octet 2
Value part								octet 3... octet n

Table 2.4: AVP example (Grouped type)

Name		Identifing header	Description
Example-AVP	::=	<AVP header: Identifier of the AVP>	Header
		{ AVP1 }	Child AVP1 mandatory
		1*{AVP2}	1 (or more) child AVP2 mandatory
		*[AVP3]	0 (or more) child AV3 optional
		[AVP4]	child AVP4 optional
		*[AVP]	Additional AVP of various types can optionally be appended

When an IE occurs in a NAS protocol message, it is flagged as either mandatory (M), optional (O) or conditional (C) in that message. The condition which determines the existence of a conditional IE in a message may only be based on an indicator in the message itself, no outer state machine should indicate this. See Table 2.3 for the general structure of an IE.

2.3.2 S6a interface

The EPS AKA signalling between the MME and the HSS is included in the S6a interface protocols. The format of the messages in these protocols are defined in [23]. Specifically, the messages 2 and 3 described in Section 2.2.3 are included in the S6a protocol – the AIR and the AIA messages. The S6a protocol is based on Diameter [24]. A Diameter message consists of a header followed by one or more attribute-value pairs (AVPs).

Attribute-value pair

An attribute-value pair (AVP) in the S6a protocol is analogous to an IE in the NAS protocol. It consists of a header which encapsulates data or possibly one or more AVPs. The format of the data field of an AVP must be one of the base data types defined for Diameter in [24] or a data type derived from those base data types [24]. The ones mainly used in this thesis are the Unsigned32, OctetString and Grouped types. A Grouped type AVP is simply an AVP which holds more AVPs. See Table 2.4 for the general structure of an AVP.

2.3.3 USIM

The USIM application is described in [25]. This includes specifications of command signatures and file formats used in the USIM and the UICC. The main points of interest in the USIM are firstly the authentication command in the USIM application and secondly storage of resources such as security keys.

Authenticate command

The USIM has a command called AUTHENTICATE. This command is used after the UE has received the Authentication Request. The command uses functions f1 to f5 as described in Section 2.2.3 in order to produce the parameters RES, CK and IK. It takes as input the RAND, SQN exclusive or AK, AMF and MAC, which are all described in Table 2.1. First the command computes AK by using function f5 and then it retrieves SQN by XORing with (SQN XOR AK). Thereafter it produces XMAC using function f1. If the received MAC is different from XMAC authentication failure occurs. Afterwards the command checks if SQN is in the correct range. If it is, the command computes RES using f2, CK using f3 and lastly IK using f4.

Storage in the UICC with elementary files (EFs)

Storage of files in the UICC is done in a file format called elementary files (EFs). The files stored include parameters IMSI, RAND, K_{ASME} and the long-term key **K**.

Chapter 3

Related work

We examine here six proposed group-based AKA protocols. For each of the protocols we explain its general idea, give an overview of its inner workings and assess its feasibility of being implemented in current mobile telecommunications systems.

3.1 Broustis et al. schemes

Broustis et al. propose three group-based authentication schemes in [7], published in 2012. All three schemes use a dedicated gateway node between the MTC devices and the network.

The security mechanism behind the schemes is based on a global challenge and on a group key which is placed differently depending on which scheme is used. What is meant by a global challenge is that for every group-based authentication, every device in the group receives an identical random challenge. The challenge is accompanied with an authentication token, produced with the group key. The token is verified either in the gateway node or in each MTC device depending on which scheme is used.

In the first scheme, it is the gateway node that holds the group key and is responsible for verifying the authentication token. If the verification is successful, the gateway broadcasts the global challenge to the MTC devices. Upon receiving the challenge, the devices use individual keys, shared with the network, to compute individual response values and session keys. They then send the responses via the gateway node to the MME, which uses the responses to authenticate the devices.

In the second and the third scheme, each MTC device (and not the gateway) holds the group key and verifies the authentication token. As for the difference between the second and the third scheme, it lies in where the responses from the MTC devices are verified. In the second scheme they are verified by the gateway, in the third by the MME.

We assess that an advantage with these schemes is that they do not introduce any new cryptographic function not existing in EPS AKA. This makes it attractive for implementation since many elements in EPS could possibly be reused. However, a disadvantage with the schemes is that a gateway node is present and adding the gateway node to the EPS architecture could prove cumbersome. Another disadvantage is that little signalling is saved between the HSS and the MME because the expected response parameters for every device must be sent for every group-based authentication.

3.2 GBAAM

Cao et al. propose a group-based authentication scheme named Group-Based Access Authentication for MTC (GBAAM) [9], published in 2015. GBAAM is based on identity-based cryptography. This is a form of public-key cryptography where the identity of an entity is used to construct the public key of this entity, giving the benefit that the public keys need not be distributed in advance.

The protocol has two phases, the *register phase* and the *group-based access authentication phase*. In the register phase, the MTC devices perform authenticated key agreement procedures that are similar to the EPS AKA protocol. The main difference is that the HSS additionally provides the MTC devices with private keys. Also in the register phase, the HSS provides the MME with a private key. An MTC group ID is included in the attach request, and the private keys are specific to this group.

In the second phase, MTC devices of a group simultaneously want to attach to the network, but this time the HSS will not be involved and the signalling goes via a *group leader* MTC device. The MTC devices make signatures with their private keys and send them to the group leader. The group leader then aggregates the signatures and sends the aggregated signature to the MME. The MME verifies the aggregated signature, and if the verification is successful the MME in turn produces a signature with its private key and sends this to the group leader, which then broadcasts this signature to the MTC devices in the group. Elliptic curve Diffie-Hellman data is also included in this scheme so that every MTC can agree on a session key with the MME.

We find that a disadvantage with the protocol is that it is based on using identity-based signatures which does not exist in the EPS security architecture, making it complex to implement. Even though the use of identity-based cryptography means that a PKI is not needed, public key parameters and private keys must be distributed to all the MMEs used by the group, and there should not be any geographical limitation to where a group might roam. We think that this makes the scheme impractical.

3.3 SE-AKA

Lai et al. propose a group-based authentication scheme named SE-AKA [8], published in 2013. In this scheme each MTC device holds a group key, a long-term key and a parameter called a *synchronization value*. The general idea of the protocol is that the HSS provides the MME with additional authentication information during an attachment of an MTC device that is a member of a group. The MME then uses this additional information in subsequent attachments of group members in order to authenticate them, saving signalling between the HSS and MME. More specifically, the additional information sent from the HSS to the MME includes a synchronization value for each MTC device in the attaching group. Between the UEs and the MME, elliptic curve Diffie-Hellman exchanges are performed for session keys. The protocol also offers a privacy enhancement by using a public-key infrastructure (PKI) to encrypt the IMSI in the initial attachment message.

We find that a disadvantage with the scheme, just as with the Broustis et al. schemes, is that the signalling between the MME and the HSS still contains information for every device in a group. This means that when a lot of MTC devices are attaching, this initial message could still possibly congest the network. Another disadvantage with the protocol is that it introduces a PKI; we believe this makes the scheme impractical to implement.

3.4 Choi–Choi–Lee scheme

Choi et al. propose a group-based authentication scheme in [10], published in 2014. In this protocol the session key derivations between the MME and the MTC devices in a group are based on a binary tree associated with the group. Every node in the tree is associated with a value. The value of every node except the root node is the result of applying a hash function on the value of the parent of the node. One hash function is used if it is a left child and another if it is a right child. Every MTC device in the group is assigned a leaf node in the tree, and the MTC device holds the values associated with all nodes in the tree except for the value of the node assigned to it and the values of the ancestors of this leaf node.

A core idea of the scheme is that a designated MTC device group leader is used as an intermediate between the network and the rest of the MTC devices in the group. The leader attaches to the network in a way similar to EPS AKA, but the IMSIs of all MTC devices in the group are also included in the initial attach message. The MME then fetches expected responses for each MTC in the group from the HSS. Thereafter, the network is authenticated using a MAC broadcast from the MME to the devices. The MTC devices send their responses to the leader which in turn sends these to the MME, which authenticates them. During the protocol, the MME gets included in the aforementioned binary tree. This enables each MTC device to agree on a session key with the MME by applying a hash function to all the tree values that are known by both the MTC and the MME.

We find that an advantage with this protocol is that it uses easily implemented functions and ideas. However, as is the case with the SE-AKA scheme and the Broustis et al. schemes, the HSS needs to produce and send data for each individual MTC device in the group. This would perhaps congest the network when a group contains a great number of MTC devices. We also identify a large security drawback with this scheme: if two compromised MTC devices with no common ancestors in the binary tree (except the root node) collude, they can obtain all the session keys of the other MTC devices.

Chapter 4

A lightweight group-based authentication protocol

In this chapter we describe the protocol of which we in Chapter 6 present a prototype implementation. We also summarize the results of a security verification of the protocol. The protocol was developed by SICS Swedish ICT in parallel with the thesis work. It is based on analysis of existing group-based authentication proposals, among them the ones we discussed in the last chapter, and constructed with the aim of being efficiently and easily implemented into current mobile telecommunications networks. Since the protocol was still under development during the project work, it is an early version of the protocol that is considered in this thesis. For the purpose of this thesis we simply refer to it as the *group-based AKA*.

4.1 Overview

The protocol has one thing in common with the Choi–Choi–Lee scheme: it uses perfect[1] binary inverted hash trees. Figure B.1 in Appendix B shows the structure of such an inverted hash tree, using the terminology that will be described in this section. The trees are built from the root downwards by using two different hash functions called h_0 and h_1 on a parent node and defining that the result from one of the functions represents the left child and the result from the other represents the right child. More specifically, the functions *h0* and *h1* and the root value are used to construct the tree in the way specified by the following recursive definition[2], where n_{ij} denotes the node at the i^{th} position and the j^{th} level.

$$n_{ij} = \begin{cases} h_0(n_{k(j-1)}) & \text{if } i = 2k & \text{(left children)} \\ h_1(n_{k(j-1)}) & \text{if } i = 2k+1 & \text{(right children)} \\ \text{given value} & \text{if } i = j = 0 & \text{(root)} \end{cases}$$

Two such trees are used in the protocol for every group. The basic idea is to use the leaf values as authentication parameters and to assign a leaf position, valid

[1]A perfect binary tree is a binary tree in which all leaf nodes are on the same level.

[2]Taken with permission from researchers at SICS.

for both trees, to each MTC device. One of the trees holds keys, called group keys, and the other holds challenges. Before any authentication and key agreement has been performed, the home subscriber server (HSS) has access to the root nodes of the trees, whereas each MTC device has access only to a value that is the result of obfuscating the group key of the device.

The general idea is that the HSS can provide an MME with sub-root nodes of these trees during an attach request for a single MTC device. The MME can then from these sub-root nodes derive authentication parameters for subsequent attachments of MTC devices that are members of the group. This allows the MME to authenticate devices without having to contact the HSS.

4.2 Further description

The two trees are called the *CH tree* and the *GK tree*. The CH tree is the tree that holds challenges and the GK tree is the tree that holds group keys. The nodes in the trees at the i^{th} position and the j^{th} level are denoted by GK_{ij} and CH_{ij}. The leaf nodes coupled with an MTC device are denoted by GK_{MTC} and CH_{MTC}.

At registration time, the network operator of the group – which must be same as the network operator of the MTC devices – provides each MTC device with a long-term key **K**, as is also done in EPS AKA. The network operator moreover provides each MTC device with a value that is produced with **K**, GK_{MTC} and CH_{MTC}. More specifically, this value, called O_{MTC}, is produced in the following way:

$$\text{O}_{\text{MTC}} = hash(\mathbf{K}, \text{CH}_{\text{MTC}}) \oplus \text{GK}_{\text{MTC}}$$

O_{MTC} is thus an obfuscation of the group key, obfuscated with **K** and CH_{MTC}. The reason behind the obfuscation is to bind the long-term key **K** and the challenge CH_{MTC} to the group key GK_{MTC}, preventing GK_{MTC} from being compromised after distribution. For instance, two corrupted MTC devices cannot swap their group keys in order to break authentication because they cannot retrieve GK_{MTC} without both **K** and CH_{MTC}.

The network operator at registration time additionally provides the MTC devices with an ID that globally uniquely identifies the group, called GID.

When a device wants to connect to the network it sends an *Attach Request* to the MME. This message contains the GID, a random number called NONCE and the tree position of the device, called PATH. On receiving the Attach Request, the MME checks whether it stores ancestor nodes of the leaf nodes defined by GID and PATH. If the answer is no, Case A is performed, and if the answer is yes, Case B is performed. We will now describe the two cases separately.

Case A

The MME sends to the HSS an *Authentication Data Request* containing the parameters received from the Attach Request (GID, PATH and NONCE) and an ID of the serving network (SN_{ID}). On receiving the Authentication Data Request, the HSS first checks whether GID and PATH are valid and whether it has never received an

Authentication Data Request for this device previously. If both conditions are met, the HSS then does two things. The first thing is that it finds IMSI and **K** associated with GID and PATH and then uses **K** and $\mathrm{SN_{ID}}$ to do the same derivations as is done by the HSS in EPS AKA. The other thing is that it, according to a policy decided by the operator of the group, selects indices i and j for nodes $\mathrm{CH_{ij}}$ and $\mathrm{GK_{ij}}$ that are ancestor nodes of the nodes defined by GID and PATH (i.e. $\mathrm{CH_{MTC}}$ and $\mathrm{GK_{MTC}}$). The policy might be based on a trade-off between security and efficiency considerations or a division of the group into smaller groups. Next, the HSS sends to the MME an *Authentication Data Response* containing the parameters included in the corresponding EPS AKA message and, additionally, CHij, GKij, GID, PATH and IMSI. The MME stores $\mathrm{CH_{ij}}$ and $\mathrm{GK_{ij}}$ for future use with other devices and mutually authenticates and agree on a key with the device in the same way as is done in EPS AKA.

Case B

The MME first checks whether it has performed a group-based AKA run with the device previously. If not, the MME then computes $\mathrm{CH_{MTC}}$ and $\mathrm{GK_{MTC}}$ by using PATH and the stored ancestor nodes $\mathrm{CH_{ij}}$ and $\mathrm{GK_{ij}}$. These derivations are done by iteratively applying two hash functions in the way described in Section 4.1. In addition to this an authentication token parameter, named AUT_D, is calculated as follows.

$$\mathrm{AUT_D} = (f5(\mathrm{GK_{MTC}},\mathrm{NONCE}), MAC_{GK_{MTC}}(\mathrm{NONCE}, \mathrm{CH_{MTC}}, \mathrm{GID}, \mathrm{SN_{ID}}, \mathrm{PATH})).$$

The MME then generates a new message named the *Authentication Request Derivable*. This message includes the parameters $\mathrm{SN_{ID}}$, $\mathrm{CH_{MTC}}$ and $\mathrm{AUT_D}$ and is sent to the MTC device.

Upon receiving the Authentication Request Derivable message, the MTC uses the $\mathrm{CH_{MTC}}$ parameter and the long term key **K** in order to de-obfuscate the $\mathrm{O_{MTC}}$ that it has stored. This is done as follows.

$$\mathrm{GK_{MTC}} = hash(\mathbf{K}, \mathrm{CH_{MTC}}) \oplus \mathrm{O_{MTC}}$$

The MTC device proceeds by verifying the MAC inside the $\mathrm{AUT_D}$. If it is correct then the network has been authenticated. Thereafter the MTC generates a new message named *Authentication Response Derivable* that contains a parameter called RES_D. It is generated as follows.

$$\mathrm{RES_D} = f2(\mathrm{GK_{MTC}}, \mathrm{CH_{MTC}})$$

This message is sent to the MME which computes a parameter $\mathrm{XRES_D}$ in the same way as $\mathrm{RES_D}$ was computed, using the stored $\mathrm{GK_{MTC}}$ and $\mathrm{CH_{MTC}}$. $\mathrm{XRES_D}$ is then compared with the received $\mathrm{RES_D}$ and if they are the same the MTC device is authenticated.

At this point the MTC device and the MME have authenticated themselves to each other and they can now compute a shared session key named K_{asmeD}. $\mathrm{K_{asmeD}}$ is generated as follows.

Table 4.1: Description of the terms introduced in the group-based AKA. The table was provided by SICS.

Term	**Description**
GID	Group identifier
PATH	The path assigned to the MTC. Each MTC is assigned with the same path in both trees.
NONCE	Random number
GK_{ij}	The (sub)group key assigned to the i-th subroot of GK tree at j-th level.
CH_{ij}	The challenge key assigned to the i-th subroot of CH tree at j-th level.
GK_{MTC}	The key associated to the MTC. It is the hash value of the leaf of the GK tree at PATH.
CH_{MTC}	The challenge key associated the MTC. It is the hash value of the leaf of the CH tree at PATH.
O_{MTC}	The obfuscated value that hides the key associated to the MTC
AUT_D	The authentication parameter in the group authentication
RES_D	The response parameter in the group authentication
K_{asmeD}	The session key generated in the group authentication

$K_{asmeD} = kdf(f5(GK_{MTC},NONCE), f3(GK_{MTC},CH_{MTC}), f4(GK_{MTC},CH_{MTC}),SN_{ID})$.

See Table 4.1 for a listing of the terms introduced by the new protocol. Figure 4.1 and Figure 4.2 illustrate the signalling used.

4.3 Security of the protocol

As stated in Chapter 1, analysis of the security of the protocol is outside the scope of the thesis. Nevertheless, we will in this section briefly summarize the results of a *ProVerif* verification that has been carried out by our supervisor Rosario Giustolisi. ProVerif is a software tool that can be used to automatically verify the security of cryptographic protocols [26]. The verification is at the time of thesis submission not published in any paper but the code that specifies the protocol with the input language of ProVerif can be seen in Appendix D.

The ProVerif code is based on an updated version of the protocol. In this version, additional hashing of the group key leaf nodes is performed in order to enable re-authentications. Although the two versions differ, the difference in terms of security is probably small or non-existent, so the results are probably valid for both versions.

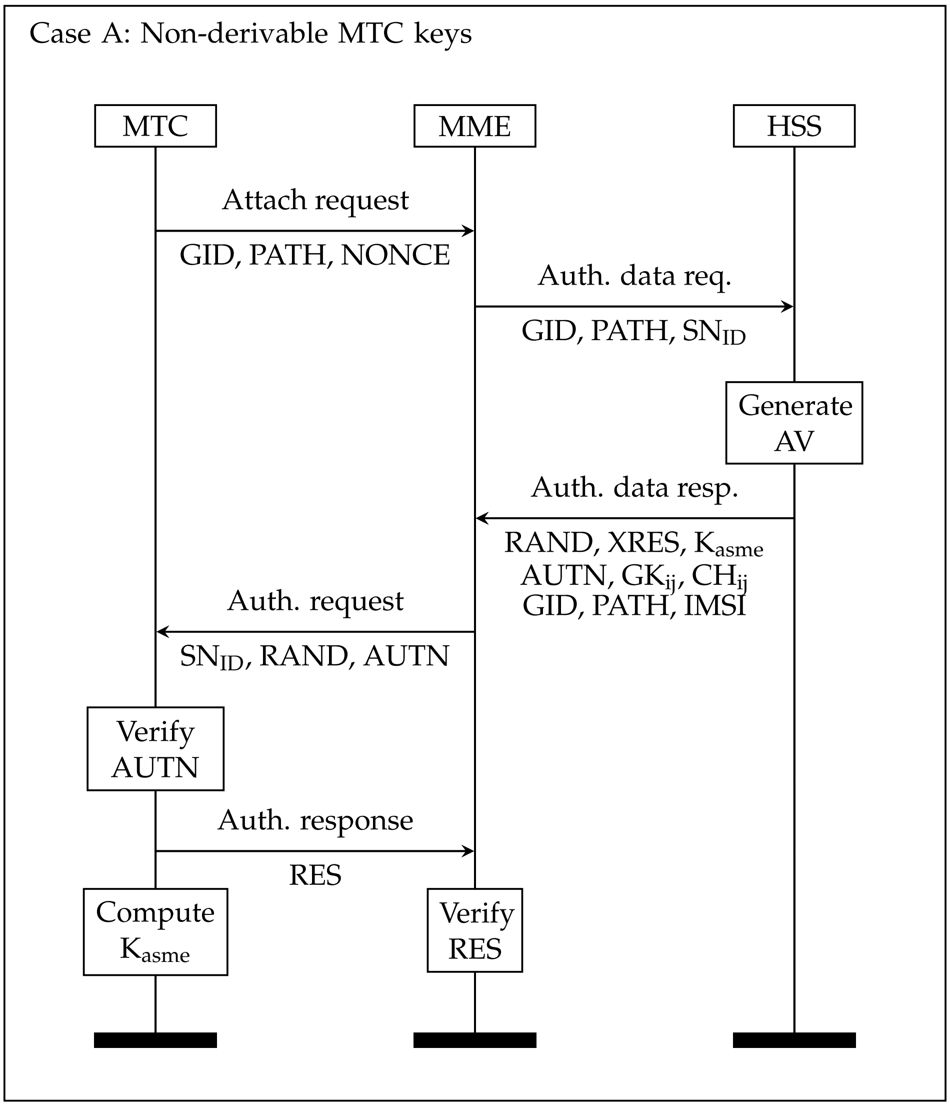


Figure 4.1: The message sequence chart of the proposed group-based AKA when the MME cannot derive the keys for the MTC device. The chart was provided by SICS.

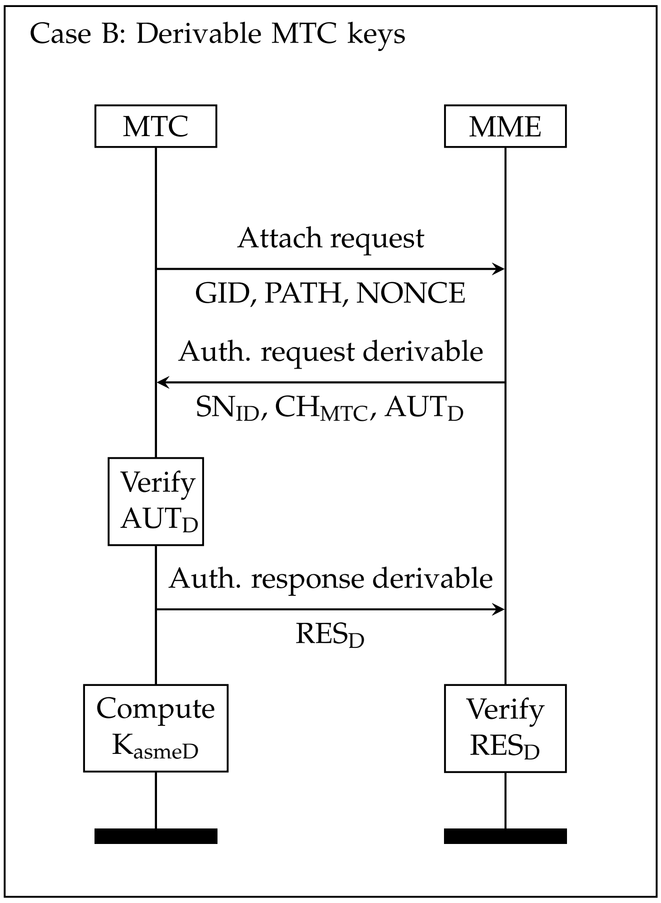


Figure 4.2: The message sequence chart of the proposed group-based AKA when the MME can derive the keys for the MTC device. The chart was provided by SICS.

The ProVerif code in Appendix D.2 considers the mutual authentication between MTC devices and the serving network. The code in Appendix D.1 considers the same scenario but includes corrupted MTC devices. Lastly, the code in Appendix D.3 considers the privacy of the MTC device identity when using the new protocol.

Preliminary results using the ProVerif code in Appendix D have shown that the protocol is secure regarding four aspects: session master key confidentiality, serving network authentication, MTC device authentication and MTC identity privacy.

Chapter 5

Tools and methodology

We evaluated eight open-source platforms that simulated the Evolved Packet System (EPS). The goal was to modify a suitable EPS platform so that it implemented the group-based AKA in a realistic environment.

In this chapter we present the evaluation and describe the platform that we chose. Lastly, we state the approach we took to the project work that remained after we had chosen a platform.

5.1 Evaluation of EPS platforms

This section is structured as follows: Subsection 5.1.1 gives the requirements for the platform; Subsection 5.1.2 gives the evaluation of how the different platforms meet the requirements; and Subsection 5.1.3 gives the reasoning behind the final choice of platform.

5.1.1 Requirements

Codebase

The codebase of the platform should be open source so that we may publish code modifications. It should also be stable and easy to understand, not least by being well structured and extensively documented. Moreover, it should be supported by an active community which can provide support. Furthermore, the codebase should be implemented in a programming language that supports good cryptographic libraries.

Realism

The platform should offer a high degree of realism with regards to the internal structure of EPS. In particular, the platform should simulate or emulate the MME, the HSS, the UE, the USIM application and the UICC. Moreover, in order to enable modification of functionality related to authentication and key agreement, the platform should support the EPS AKA protocol. This includes NAS signalling, the Diameter protocol, the security functions f1–f5 and the USIM AUTHENTICATE command.

Traffic measurements

The platform should be able to log communication channels between certain entities in EPS, in order to provide a basis for signalling tests. In particular, the platform should be able to log the AKA exchanges between the UE, the MME and the HSS. This would entail that the following EPS interfaces can be logged.

- S6a interface (MME – HSS) for the Diameter protocol
- S1-MME (eNodeB – MME) for NAS protocols signalling

Security measurements

It is good if the platform supports some form of security analysis tool, so that an analysis of the security of the new protocol might be facilitated.

Computational measurements

It is good if the platform supports some form of computational analysis tool, so that an analysis of the performance of the new protocol might be facilitated.

MTC devices

Regarding the number of MTC devices the platform is able to simulate or emulate at the same time, it would be best if the platform could simulate a massive amount of MTC devices. However, if the platform cannot do that, the more MTC devices the platform can simulate, the better.

5.1.2 Platforms

Here we describe and evaluate each of the platforms one by one. A summary of the evaluation can be seen in Table 5.1.

LTE-Sim

LTE-Sim [27] is an event-driven simulator used for simulating uplink and downlink scheduling strategies. It is mainly used for radio interface simulation. It supports simulation of the entities UE, eNodeB and MME with regards to radio interfaces. It offers an open-source codebase written in C++. It also partially implements the core network. However, no EPS AKA protocol elements were found in the codebase and no HSS entity is supported. Hence, we do not choose to use this platform.

OpenLTE

OpenLTE [28] is an open-source radio interface simulator written in Octave, Python and C++. It is mainly used to simulate an eNodeB and to extend the capabilities of GNU Radio applications, which is an open-source toolkit for software-defined radio. The focus is on transmission and reception of downlink communication from an eNodeB to a UE. This platform does include certain elements of interest as it has partly implemented an HSS and an MME. Moreover, there exist parts of the EPS AKA protocol in the codebase. However, it cannot simulate the actual radio communication, instead it actually performs it. This means that in order to use this simulator, a radio transmitter and a physical UE are needed. Such a transmitter is out of scope for this thesis. Thus, we do not choose to use this platform.

srsLTE + srsUE

srsLTE [29], written in C, and srsUE [30], written in C++, are open-source simulators used for software-defined radio. They are mainly used for testing radio interfaces. However, they do support EPS AKA to a certain extent as the functions f1–f5 are included in the codebase, and it is also stated that the UE supports the full EPS protocol stack. However, no HSS entity is provided. Another drawback with these simulators is that the radio transmission between a UE and the network is not simulated, which means that radio hardware is required. Thus, we do not choose to use this platform.

ns-3 + LTE Module

ns-3 [31] with the LTE Module [32] is a popular open-source discrete-event network simulator written in C++. It is used for a large range of purposes, for example packet scheduling, radio resource management and inter-cell interference coordination. It supports simulation of multiple UEs, eNodeBs, GW nodes and MMEs. The simulation is complete in the sense that the radio interface is also simulated. Furthermore, the platform offers extensive documentation of the code and an active community. However, the platform does not offer control signalling. In particular, the EMM protocol and the Diameter protocol are not included or grossly abstracted. We conclude that adding this logic to the simulator would be time consuming. However, since the platform offers the possibility of simulating a large amount of UE, it was considered.

OMNet++ and 4GSIM

OMNet++ and 4GSIM [33] is an open-source discrete-event simulator written in C++. It is used for 4G simulation. It simulates the eNodeB, the MME and the HSS. It does not have an active community and is missing a UE simulation, which would require some external UE simulation platform to be integrated with this platform. Thus, we do not choose to use this platform.

SimuLTE

SimuLTE [34] is a discrete-event simulator used for EPS simulation written in C++. It is mainly used for simulation of user-plane signalling. It supports a UE and an eNodeB and has an active community. It does not include EPS AKA or any control signalling and is missing key entities. Thus, we do not choose to use this platform.

Python Protocol Simulator

Python Protocol Simulator [35] is an open-source protocol simulator written in Python. It includes simulation of the Diameter protocol and an HSS entity. However, it does not implement any other entity or protocol related to the EPS. Thus, we do not choose to use this platform.

OpenAirInterface

OpenAirInterface is an open-source wireless technology platform [36] written in C. It is a fully-stacked EPS implementation with the goal of being used for development and research. It supports an MME, an HSS, an eNodeB and an abstracted UE. It does not require any radio hardware in order to run since it can simulate the radio interface used in EPS by Ethernet. It can run the EPS AKA protocol from the UE to the HSS and back. Furthermore, it does have an active community. A disadvantage with the platform is that the high ambitions of the people behind it make it complex, and since it is in rapid development it could also prove to be unstable. Moreover, it currently does not support more than a few UEs at a time.

5.1.3 Conclusion

In this evaluation a number of EPS platforms have been investigated. A comparison of the platforms is depicted in Table 5.1. From this we identify two directions that can be taken. The first direction is to use ns-3 and the LTE Module because it is able to simulate many UE devices. Also, in contrast to the 4GSim which also can simulate many MTC devices, ns-3 supports the complete user plane of the EPS network and has an active community. However, the EPS AKA protocol and in fact all control signalling is missing from ns-3. If this platform was chosen many parts would most likely be required to be built from the bottom up.

The other direction is OpenAirInterface. This platform is very complex since it offers an almost complete implementation of EPS. By using this platform, EPS AKA signalling and functions can be reused or modified. However, it cannot, currently, support more than a few UEs. This means that any analysis aspect of looking at a massive amount of MTC devices connecting might be complicated.

Regarding the other platforms, they simply do not support enough parts of EPS or they require additional hardware that cannot be obtained in accordance with the thesis scope. Therefore, they will not be considered.

We select OpenAirInterface (OAI) instead of ns-3 and the LTE module as the platform for this thesis work. The main motivation behind using OAI is that it provides a realistic basis for implementing the group-based protocol. Since the goal of the OAI project is to provide a realistic implementation of a fully-stacked core network, it follows closer to our objective of investigating and providing a prototype that is feasible to implement in current mobile telecommunication networks as stated in Section 1.1. Furthermore, OAI provides a basis on which signalling and security functions can be re-purposed or re-defined in. However, a consequence of this decision is that testing the new protocol using a massive amount of MTC devices can only be done in future work.

5.2 Details of OpenAirInterface

The goal of the OAI wireless technology platform is to enable an open EPS ecosystem released as free software under the OSA license model [1]. It was founded by EURECOM [2] together with a number of supporting companies. The platform offers an open-source implementation of the EPS radio interface, EUTRAN, and the core network. This also includes NAS integrity and encryption using AES, a popular encryption algorithm, and SNOW 3G, a stream cipher used by 3GPP as encryption algorithm. Furthermore, it can handle attach, authentication, service access and radio bearer establishment. In particular, the NAS protocols and the S6a interface mentioned in Section 2.3 are implemented in the codebase. OAI offers a platform on which researchers and companies can develop EPS and 5G deployment solutions and experimentation.

OAI is also able to simulate an eNodeB and a UE that can connect to the core network with a simulated radio interface. However, this is currently under early development so certain aspects are abstracted in this scenario. In particular, the UE is simulated in such a way that the USIM application file system is non-existent, instead a simple file is written to and read from the host. The USIM does have the security functions implemented (AUTHENTICATE and f1–f5), which are the main functions needed for remodelling in the implementation of the group-based AKA protocol.

OAI is divided into two projects: Openair-cn and Openairinterface5G.

5.2.1 Openair-cn

Openair-cn consists of the MME and the HSS entities and the protocol stack running on top of them. In particular, the MME and HSS are run as their own processes and each major protocol or procedure is run as threads in these processes. The threads communicate to each other within an MME or an HSS by using something called the inter-task interface, which is a scheduler for message passing.

HSS

The HSS process consists of an S6a thread and a database. The database application used is MySQL: it represents the AuC and stores user authentication parameters. The HSS also includes an authentication and key derivation codebase for the KDF and the f0 function. The lower-level cryptographic functions, such as an HMAC

[1]OSA Public License V1.0 and the Apache V2.0 License.

[2]EURECOM is a graduate school and research centre in communication systems located in France. Website: http://www.eurecom.fr/en.

Table 5.1: Comparison of EPS platforms

	Entities			Signalling		Logging	EPS AKA			MTC	HW req.	Progr. language
	UE	MME	HSS	S6a	NAS (EMM)		f1–f5 & KDF	AUTH CMD	Full support	#		
LTE-SIM	Y	Y	N	N	Partly	Y	N	N	N	Atleast 1	N	C++
OpenLTE	N	Partly	Partly	Partly	Partly	Y	Partly	Partly	N	Atleast 1	Y	Octave, Python & C++
srsLTE + srsUE	Y	Y	N	N	Y	Y	Y(USIM)	Y	N	Atleast 1	Y	C & C++
ns-3 + LTE Module	Y	Y	Partly	N	Partly	Y	N	N	N	Lots	N	C++
OMNet++ and 4GSIM	N	N	N	Y	Y	Y	N	N	N	Lots	N	C++
SimuLTE	Y	N	N	N	N	Y	N	N	N	Atleast 1	N	C++
Python protocol simulator	N	N	Y	Y	N	Y	Partly	Y	N	0	N	Python
OpenAirInterface	Y	Y	Y	Y	Y	Y	Y	Y	Y	Atleast 1	N	C

used in the system, stem from an open-source library called Nettle [37], which is included in the HSS.

An open-source Diameter implementation called freeDiameter [38] is used for S6a signalling. During the building of the HSS and the MME, the freeDiameter source code is fetched and subsequently patched for the purpose of introducing an S6a extension.

MME

The MME process runs several threads, where the ones of main interest are the NAS protocols thread and the S6a thread. Just like with the HSS, the S6a thread uses freeDiameter to send and receive Diameter messages in the S6A interface.

5.2.2 Openairinterface5g

Openairinterface5G is responsible for the eNodeB and UE simulation in this setup. It consists of an abstracted UE and eNodeB combination that provides realistic radio stack signalling and NAS signalling when connected to the Openair-cn.

5.3 Approach

Our main task was to translate the group-based AKA from the high-level description seen in Chapter 4 to a specification and implementation as seen in Chapter 6. The purpose was to showcase the feasibility of the protocol. We reasoned that the best way to do this was by developing the protocol on the realistic EPS platform OAI and taking into account the existing 3GPP documentation surrounding EPS AKA.

The approach for the implementation was centred around evolving the EPS AKA protocol so that the group-based AKA could be supported by EPS. A first step to this was to identify the signalling differences between classic AKA and the group-based AKA. From this information a set of new messages were specified for the protocol which were based on EPS AKA. The next step in the approach was to identify the size, structure and function of any new parameter introduced in the group-based AKA. The choice of the sizes and structure of the parameters took into account how they interact with the classic EPS AKA functionality and how they could fit into classic EPS AKA signalling so that the realistic property of OAI could be retained. This approach would inherently enable detection of any backwards compatibility issues.

Our secondary task was to analyse the performance of the protocol. The purpose of the analysis was to evaluate how the group-based AKA protocol fares against the EPS AKA protocol and draw a comparison between them. This analysis is meant to further the question of the feasibility of the protocol. The analysis was based around the scenario of a lot of MTC devices attaching to the network and the properties of interest in the analysis were latency and traffic size.

The analysis had both empirical and theoretical components. For the empirical components we used the code that we wrote and the already existing OAI code.

The same configuration was used for the tests as for when we ran the implementation when we tested the general feasibility of the implementation. Wireshark and ping provided us with measurement data. For the theoretical components, simplifications were made. However, the aim was to motivate every choice we had to make.

Chapter 6

Our implementation

In Chapter 4 we described the group-based AKA in broad terms. In this chapter we provide a detailed specification that aims to serve as a reference for a possible standardization of the protocol. The guiding principles for the design choices we make in this chapter is that the security should be good, the performance should be good and the implementation should be as practical as possible in every possible respect, for example in terms of backwards compatibility.

6.1 MILENAGE

With EPS AKA, the cryptographic algorithms used for authentication and key agreement are not standardized. The reason behind this is that the same organisation controls both the Universal Integrated Circuit Card (UICC) and the Home Subscriber Server (HSS), which is the places where the algorithms are executed. The situation looks entirely different with the new group-based AKA protocol: the cryptographic algorithms that are executed on the network are executed at the Mobility Management Entity (MME) and not at the HSS. Consequentially, the algorithms must be standardized. Since our approach, which we described in the last chapter, is to stay as close to the 3rd Generation Partnership Project (3GPP) specifications as possible, we decide to take the MILENAGE definitions of the functions f2–f5 as a point of departure for the specifications of our versions of the functions. The kernel function specified for MILENAGE, Rijndael/AES, is also specified by us for the functions $\mathrm{MAC}_{\mathrm{GK}_{\mathrm{MTC}}}$ and f2–f5.

6.2 New parameters

The parameters we introduce in this section are based on the parameters of the new protocol which can be seen in Table 4.1 in Chapter 4.

GID

When a serving network receives an attach request, it must somehow deduce to which home network the device belongs, so that it can send a request for authentication information. In EPS AKA the serving network identifies which mobile network operator, and hence which home network, the device belongs to by looking at two fields in the international mobile subscriber identity (IMSI): the mobile country code (MCC) and the mobile network code (MNC). In the group-based AKA the IMSI is not sent in the attach request, but instead GID, PATH and NONCE is sent. One of these parameters should take the place of IMSI in the sense that it provides the serving network with the previously mentioned mobile network operator identifier. The only one of the three new parameters for which it makes any sense to include this identifier is the GID. More generally, the function of GID is similar to the function of IMSI: it is an identifier which partially identifies which AKA parameters that is paired with the device.

Therefore, we design the GID just like the IMSI is designed. An advantage with this solution is that it allows current implementations to reuse the structure of IMSI in storage and processing. Furthermore, the variable size of the parameter allows any implementer to construct any size up to 15 digits based on their preference. The structure of GID can be seen in Figure 6.1.

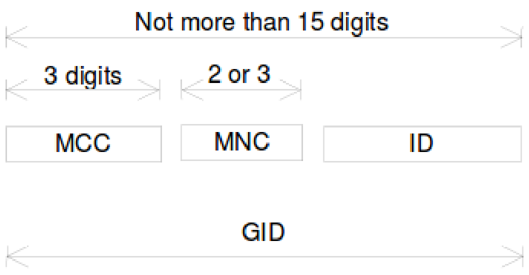


Figure 6.1: The structure of GID.

PATH

Regarding the length of the PATH, it is difficult to assess what length would be suitable since it depends on how large the largest groups will typically be and whether the division of a group tree into sub trees will entail that a lot of PATH values will have to be unused. Nevertheless, since we have to make a decision, we decide that the the length of PATH is 1 to 32 bytes. With this length range, a group can consist of up to 2^{256} members.

NONCE

In EPS AKA, f5 is a function that is applied on the parameter RAND and yields the anonymity key (AK) [20]. In the group-based AKA, the input to f5 is not RAND but NONCE. To meet the existing requirements of f5, NONCE must thus have the same size as RAND. Therefore, we decide that NONCE consists of 16 bytes.

GK_{ij}, CH_{ij}, GK_{MTC}, CH_{MTC} and O_{MTC}

The hash functions h_0 and h_1 that are used to compute the hash values of a node's left and right child, respectively, have to output 128 bits since this is the key length used in the functions f2, f3, f4 and f5, and the length of the other, non-key, input parameters to the functions f2, f3 and f4. Consequently, 128 bits must be the length of the parameters GK_{ij}, CH_{ij}, GK_{MTC}, CH_{MTC} and O_{MTC}. This length allows an implementer to reuse the current structure and functions in the EPS.

AUT_D

We specify here the size of the authentication parameter AUT_D. It contains a MAC value which we specify as 64 bits. The reason for this is that the function which computes the MAC in the classic EPS AKA also outputs 64 bits. The AUT_D parameter also consists of the data *f5*(GK_{MTC}, NONCE) which we specify is 48 bits. The size of this parameter is based on the fact that the f5 function outputs 48 bits. This structure was chosen so that it resembles the parameter AUTN in EPS AKA. For the purpose of this thesis we denote the parameter *f5*(GK_{MTC}, NONCE) by TEMP from now on.

RES_D

Because RES_D is the output of the function f2 and the output of this function as specified in the MILENAGE algorithm set is 8 bytes, we specify that RES_D is also 8 bytes in size.

Auxiliary parameters: TREE HEIGHT and NODE DEPTH

We introduce two auxiliary parameters which are related to the PATH parameter and are needed for practical reasons.

The first is called TREE HEIGHT and gives the height of the inverted hash trees. It is used as an indicator of how many bits in the PATH parameter that should be used when giving the path in the inverted hash trees. The parameter is needed because PATH must be communicated in full bytes even though the size of the actual PATH does not have to be divisible by eight. We specify that the size of TREE HEIGHT is one byte.

The second is called NODE DEPTH and gives the level on which the sub-root nodes GK_{ij} and CH_{ij} are placed in the inverted hash trees. It is used in the MME during Case B of the group-based AKA protocol. When the MME receives GID and PATH from the MTC device, the NODE DEPTH is needed in order to correctly interpret which bits in PATH signifies the path to take in the inverted hash trees from the sub-root nodes GK_{ij} and CH_{ij}.

6.3 Messages

In this section we describe our implementation of the messages sent in the group-based AKA. There are in total seven of them, but two of them, the last two sent in Case A, are identical to the corresponding messages in EPS AKA. A design principle that we follow when we design the group-based AKA messages is to base them on the EPS AKA messages and only make changes to the messages that are required by the changes that the new protocol introduces. Therefore, every message in the group-based AKA has a corresponding message in EPS AKA. The EPS AKA messages can be viewed in Appendix A.

We distinguish between three different ways in which we can base a message on an existing message. The first is that we use the existing definition of the message, including which information elements (IEs) are used, in which order the IEs must be coded, etc. The changes in this case are limited to the meaning that the parameters have and possibly a restriction of the size of some IE. The second is that we modify the definition of the message without changing the name and the type identifier of the message. The third is that we change the type identifier of the message and adds the word derivable to the end of the name of the message. In this case there may, of course, also be other changes to the definition of the message. The five new messages in the group-based AKA protocol are as follows.

6.3.1 Attach Request

For the message called Attach Request in Chapter 4, we modify the EPS AKA message that is also called Attach Request, which has the structure shown in Table A.1 in Appendix A. The differences between the EPS AKA version and this new version are firstly that a GID is sent instead of an IMSI, a GUTI, or an IMEI, and secondly that the additional parameters PATH and NONCE are included in the message. As mentioned in Section 2.3.1, this signalling is done through L3 protocols, so IEs are used in the description of the message. The composition of the message can be seen in Table 6.1. The IEs called PATH IE and NONCE IE are introduced by us. PATH IE contains the PATH and NONCE IE contains the NONCE.

The original Attach Request message includes an EPS mobile identity IE [22], which is an IE that contains the UE identity parameter: either the IMSI, the GUTI or the IMEI. In order to identify which one of these three types of identity that is used, an EPS mobile identity type value is included in this IE. We choose to extend the EPS mobile identity IE to also support the GID. The new type value is specified as follows.

Bits	3	2	1	
	1	0	1	GID

Note that the presence requirements of the PATH IE and the NONCE IE are set to conditional (C) in Table 6.1. The condition for both of them is whether the GID parameter is present or not in the EPS mobility identity IE. The idea is that if the type is GID in this IE, then this indicates that the PATH IE and the NONCE IE are also present in the message. It also indicates to the MME that the group-based AKA should be run rather than EPS AKA.

Table 6.1: Modified ATTACH REQUEST message content

IEI	Information Element	Type/Reference [22]	Presence	Format	Length
	Protocol discriminator	Protocol discriminator 9.2	M	V	1/2
	Security header type	Security header type 9.3.1	M	V	1/2
	Attach request message identity	Message type 9.8	M	V	1
	EPS attach type	EPS attach type 9.9.3.11	M	V	1/2
	NAS key set identifier	NAS key set identifier 9.9.3.21	M	V	1/2
	EPS mobile identity	**EPS mobile identity** 9.9.3.12	**M**	**LV**	**5-12**
	UE network capability	UE network capability 9.9.3.34	M	LV	3-14
	ESM message container	ESM message container 9.9.3.15	M	LV-E	5-n
19	Old P-TMSI signature	P-TMSI signature 10.5.5.8	O	TV	4
50	Additional GUTI	EPS mobile identity 9.9.3.12	O	TLV	13
52	Last visited registe TAI	Tracking area identity 9.9.3.32	O	TV	6
5C	DRX parameter	DRX parameter 9.9.3.8	O	TV	3
31	MS network capability	MS network capability 9.9.3.20	O	TLV	4-10
13	Old location area identification	Location area identification 9.9.2.2	O	TV	6
9-	TMSI status	TMSI status 9.9.3.31	O	TV	1
11	Mobile station classmark 2	Mobile station classmark 2 9.9.2.4	O	TLV	5
20	Mobile station classmark 3	Mobile station classmark 3 9.9.2.5	O	TLV	2-34
40	Supported Codecs	Supported Codec List 9.9.2.10	O	TLV	5-n
F-	Additional update type	Additional update type 9.9.3.0B	O	TV	1
5D	Voice domain preference and UE's usage setting	Voice domain preference and UE's usage setting 9.9.3.44	O	TLV	3
D-	Device properties	Device properties 9.9.2.0A	O	TV	1
E-	Old GUTI type	GUTI type 9.9.3.45	O	TV	1
C-	MS network feature support	MS network feature support 9.9.3.20A	O	TV	1
–	**PATH IE**	**PATH IE**	**C**	**LV**	**2-33**
–	**NONCE IE**	**NONCE IE**	**C**	**V**	**16**

6.3.2 Authentication Information Request

For the message called Authentication Data Request in Chapter 4, we modify the EPS AKA message Authentication Information Request, which has the structure shown in Table A.4 in Appendix A. The changes between the old and the new versions are firstly that a GID is sent instead of an IMSI and secondly that the parameter PATH is included. As mentioned in Section 2.3.2, this signalling is done with Diameter, so AVPs are used in the description of the message. The message structure is as follows.

```
<Authentication-Information-Request>  ::=  <Diameter Header: 318, REQ, PXY, 16777251 >
                                           <Session-Id >
                                           [ Vendor-Specific-Application-Id ]
                                           { Auth-Session-State }
                                           { Origin-Host }
                                           { Origin-Realm }
                                           [ Destination-Host ]
                                           { Destination-Realm }
                                           { User-Name }
                                           *[Supported-Features]
                                           [ Requested-EUTRAN-Authentication-Info ]
                                           [ Requested-UTRAN-GERAN-Authentication-Info ]
                                           { Visited-PLMN-Id }
                                           *[ AVP ]
                                           *[ Proxy-Info ]
                                           *[ Route-Record ]
```

The message includes a User-Name AVP, which is used for sending user-names in Diameter. In EPS AKA the User-Name AVP contains an IMSI, but for the new protocol we specify that the GID parameter can also be included in this AVP.

The message also includes an AVP named Requested-EUTRAN-Authentication-Info which shall, according to [23], contain information related to authentication requests for Evolved Universal Terrestrial Radio Access Network (E-UTRAN). Since the parameter PATH is related to authentication procedures for E-UTRAN services we introduce the PATH as an optional AVP to the Requested-EUTRAN-Authentication-Info AVP. The modified Requested-EUTRAN-Authentication-Info AVP is as follows.

```
Requested- EUTRAN-Authentication-Info  ::=  <AVP header: 1408 10415>
                                            [ Number-Of-Requested-Vectors]
                                            [Immediate-Response-Preferred ]
                                            [ Re-synchronization-Info ]
                                            [ PATH AVP]
                                            *[AVP]
```

The presence of the new PATH AVP indicates that the new protocol is run and consequently that the User-Name AVP holds a GID and not an IMSI.

According to the new protocol, SN_{ID} should also be sent in this message, but this must also be sent in EPS AKA, so the inclusion of SN_{ID} does not necessitate any additional change of the message.

6.3.3 Authentication Information Answer

For the message called Authentication Data Answer in Chapter 4, we modify the EPS AKA message Authentication Information Answer, which can be seen in Table A.5 in Appendix A. The difference between the two is that the parameters CH_{ij}, GK_{ij}, GID, PATH, IMSI, TREE HEIGHT and NODE DEPTH are also included in the modified message. As mentioned in Section 2.3.2, this signalling is done through Diameter, so AVPs are used in the description of the message. The message structure is as follows.

<Authentication-Information-Answer> ::= <Diameter Header: 318, PXY, 16777251>
<Session-Id >
[Vendor-Specific-Application-Id]
[Result-Code]
[Experimental-Result]
[Error-Diagnostic]
{ Auth-Session-State }
{ Origin-Host }
{ Origin-Realm }
* [Supported-Features]
[Authentication-Info]
*[AVP]
*[Failed-AVP]
*[Proxy-Info]
*[Route-Record]

The message includes an AVP named Authentication-Info which in turn includes authentication vector AVPs. One such vector is the E-UTRAN-Vector AVP which consists of the EPS AKA authentication parameters. The AVP is as follows.

Authentication-Info ::= <AVP header: 1413 10415>
*[E-UTRAN-Vector]
*[UTRAN-Vector]
*[GERAN-Vector]
*[AVP]

We choose to construct a new such vector including the aforementioned group-based AKA parameters. This new vector is then appended to the Authentication-Info AVP. The new Group-authentication-vector is as follows.

Group-auth-vector ::= <AVP header: >
[Item-Number]
{ NODE DEPTH }
{ TREE HEIGHT }
{ GK_{ij} }
{ CH_{ij} }
2{User-Name}
{PATH}
*[AVP]

Table 6.2: AUTHENTICATION REQUEST DERIVABLE message content

IEI	Information Element	Type/Reference	Presence	Format	Length
	Protocol discriminator	Protocol discriminator 9.2	M	V	1/2
	Security header type	Security header type 9.3.1	M	V	1/2
	Authentication request derivable message type	Message type 9.8	M	V	1
	NAS key set identifier $_{\text{ASME}}$	NAS key set identifier 9.9.3.21	M	V	1/2
	Spare half octet	Spare half octet 9.9.2.9	M	V	1/2
	CH$_{\text{MTC}}$ IE	**CH$_{\text{MTC}}$ IE**	**M**	**V**	**16**
	AUT$_{\text{D}}$ IE	**AUT$_{\text{D}}$ IE**	**M**	**LV**	**15**

6.3.4 Authentication Request Derivable

For the message called Authentication Request Derivable in Chapter 4, we introduce a new message, also called Authentication Request Derivable. We base this message on the corresponding message in EPS AKA, Authentication Request, which can be seen in Table A.2 in Appendix A. Authentication Request Derivable differs from Authentication Request in that it includes the parameters CH$_{\text{MTC}}$ and AUT$_{\text{D}}$ instead of the parameters RAND and AUTN. For these parameters we introduce the IEs CH$_{\text{MTC}}$ IE and AUT$_{\text{D}}$ IE. AUT$_{\text{D}}$ is coded as the concatenation of TEMP and MAC. Table 6.2 shows the structure of Authentication Request Derivable.

When defining a new message, a new type identifier is required. This is done by specifying a new EPS mobility management message type. The new message type identifier is specified as follows.

Bits	8	7	6	5	4	3	2	1	
	0	1	0	1	0	1	1	1	Authentication Request Derivable

6.3.5 Authentication Response

For the message called Authentication Response in Chapter 4, we use the message in EPS AKA also called Authentication Response. We do not change its definition, which can be seen in Table A.3 in Appendix A. The difference in the interpretation of the parameters when the message is sent in EPS AKA and when it is sent in the group-based AKA is that in the latter case the authentication response parameter IE contains RES$_{\text{D}}$ rather than RES. As can be seen in Table A.3, the length of the authentication response parameter IE is defined to be in the range 5–17 bytes.

Note that in the group-based AKA, the IE is always 9 bytes, since we specified in Section 6.2 that RES_D consists of eight bytes and the format of the IE is LV.

6.3.6 Notes on AVP parameters

The AVP parameters introduced in the modified authentication information request and authentication information answer messages must also be specified by data types used in the Diameter protocol. We specify these as follows.

Attribute name	**Defined in**	**Value type**
PATH AVP	Section 6.2	OctetString
GK_{ij} AVP	Section 6.2	OctetString
CH_{ij} AVP	Section 6.2	OctetString
TREE HEIGHT AVP	Section 6.2	Unsigned32
NODE DEPTH AVP	Section 6.2	Unsigned32

6.3.7 Note on identifier values

Identifier values are used in the 3GPP specifications for each information element (IE) and each attribute-value pair (AVP) in order to differentiate them in the Non-Access Stratum (NAS) protocol stack and in the S6a interface, respectively. For the purpose of this thesis we specify that any new IEs or AVPs introduced in this section have values that are currently not in use by any other IE or AVP.

6.4 Functions and commands

We introduce new functions and commands that extend the functionality currently in use in EPS AKA. This can be done since the new protocol reuses many of the current security functions.

6.4.1 Assumptions

As stated in Section 6.1, we use the MILENAGE algorithm set for the functions f2–f5. However, since the subscriber operator does not necessarily control the MME, an Operator Variant Algorithm Configuration Field (OP) cannot be used. The OP and K is used to derive the value OP_c in EPS AKA, and in the specification of MILENAGE [2] this value is an input to the algorithms f2–f5. To get around this, we specify that OP_C is defined as all zeros.

6.4.2 Hash function choices

The group-based AKA protocol uses three cryptographic hash algorithms named *h0*, *h1* and *hash* as stated in Section 4.2. We choose to use the SHA-256 hash algorithm for *h0* and *h1* since SHA-256 outputs a reasonable amount of bits for the purpose of these functions. Additionally, during the writing of this thesis, SHA-256 is generally considered secure. We choose to reuse the current USIM f3 function for *hash* for reasons mentioned below.

h0 and *h1*

We specify that SHA-256 shall be used for the *h0* and *h1* hash functions. A *h0* hash, the left hash, will be done by appending a zero byte to the input CH_{ij} or GK_{ij} before using SHA-256. The *h1* hash, the right hash, will be performed by appending a byte of all ones to the input CH_{ij} or GK_{ij} before using SHA-256. The result of the hash will be truncated to 128 bits and this value represents the children node of CH_{ij} and GK_{ij}. See pseudo-code 2 for a more detailed description.

USIM: *hash*

According to clause 6.2 of [25], the cryptographic functions supported by the USIM do not include any hash function. But for the new protocol to work as specified in Chapter 4, it is required that the USIM be able to compute a hash value. Since *hash* takes the long term key **K** as input and it is not recommended that **K** leave the USIM, *hash* should not be performed outside of the USIM. A solution to this issue is to instead re-use the current USIM functions f3 or f4 for *hash*.

The f3 and f4 functions, as described in Section 2.2.2, are used to produce the CK and IK parameters. The input parameters to these functions are **K** and the 128-bit value RAND and the output is a 128-bit value. This is almost the same signature as *hash* needs for our implementation. Furthermore, the requirement of f3 and f4 as stated in clause 5.1.6.7 of [20] is that it shall be computationally infeasible to derive **K** from knowledge of RAND and the output. It is reasonable to assume that these functions would also provide the same for the parameters **K**, CH_{MTC}, and O_{MTC}, which would protect **K** in case the parameters O_{MTC} and CH_{MTC} are compromised. Moreover, since we choose static sizes for the parameter CH_{MTC} and **K** is set at 128 bits, the hash will always use the same input size and therefore lose the compression property of hash functions. Hence, reusing f3 or f4 for *hash* is acceptable and therefore we choose to use f3 for the *hash* function.

6.4.3 Reuse of f2–f5

The sizes chosen for the new parameters in Section 6.2 fit into functions f2–f5 just as the current EPS AKA parameters seen in Section 2.2.3. This means that we can produce the new parameters just as specified in Section 4.2. The way we reuse the functions f2–f5 is specified as follows.

Param.	**Produced by**
$XRES_D$	$f2(GK_{MTC}, CH_{MTC})$
CK_D	$f3(GK_{MTC}, CH_{MTC})$
IK_D	$f4(GK_{MTC}, CH_{MTC})$
TEMP	$f5(GK_{MTC}, NONCE)$
K_{asmeD}	$kdf(TEMP, CK_D, IK_D, SN_{ID})$

6.4.4 USIM and MME: f1 derivative

The new protocol requires a parameter called $MAC_{GK_{MTC}}$ to be produced in the USIM and the MME, as seen in Section 4.2. Because $MAC_{GK_{MTC}}$ is generated by a larger input than the function that computes the MAC in EPS AKA, f1, the function f1 cannot be reused. Therefore, we introduce a new MAC function which we call *f1 derivative*. This function has the following signature.

Input:	(GK_{MTC}, NONCE, CH_{MTC}, GID, SN_{ID}, PATH, nbr of bytes in GID, nbr of bytes in PATH)
Output:	($MAC_{GK_{MTC}}$)

The function is a cipher block chaining (CBC) MAC which is based on the CBC MAC function f1 used in the OAI platform. The major difference between f1 derivative and f1 is that f1 derivative has a larger input which does not have a fixed size. As already mentioned an OP_C is also not used in our implementation and therefore f1 derivative does not use it when f1 does.

Since f1 derivative has a variable size input, some new methods are introduced in order to provide the same security aspects as f1. These methods are based on the recommendations of [39] which state that variable size inputs to a CBC MAC can be securely authenticated by pre-pending the size of the input message in the first block. Moreover, since each block must be 16 bytes there is a need for padding during the situation when the input size in bytes is not divisible by 16. When padding is done it is recommended by [39] that the padding bits should be started by a 1 and then followed by 0s. We implement the f1 derivative function in accordance with these recommendations and the code for f1 derivative can be found in Appendix C.2.

6.4.5 USIM: GROUP-BASED AUTHENTICATE

In order for the USIM application to authenticate the serving network and produce RES_D we introduce a new USIM command named GROUP-BASED AUTHENTICATE. This new command is based on the AUTHENTICATE command described in Section 2.3.3. This command should be used after Authentication Request Derivable has been received by the MTC device. The command must be standardized for interoperability between ME and USIM applications, since these could be represented by different manufacturers. It has the following signature.

Input:	(CH_{MTC}, SN_{ID}, $MAC_{GK_{MTC}}$, NONCE)
Output:	(RES_D, CK, IK)

The function first computes GK_{MTC} using the CH_{MTC} and the stored O_{MTC} value as described in Section 4.2. The GK_{MTC} is now used with the CH_{MTC}, NONCE, GID, SN_{ID} together with the PATH variable in order to compute $XMAC_{GK_{MTC}}$ using the f1 derivative function. The next step will be to verify the received $MAC_{GK_{MTC}}$ by comparing it with the XMAC. If successful, the function continues by producing RES_D, CK and IK using the functions f2, f3 and f4. The output from this command together with the SN_{ID} can now be used with the KDF function in order to produce K_{asmeD}. Pseudocode for the command can be seen in pseudo-code 1.

Pseudo-code 1: USIM: GROUP-BASED AUTHENTICATE

```
input  : $CH_{MTC}$, $SN_{ID}$, $MAC_{GK_{MTC}}$, NONCE
output : $RES_D$, CK, IK

$GK_{MTC} \leftarrow$ f3($K$, $CH_{MTC}$) XOR $O_{MTC}$;

$XMAC_{GK_{MTC}} \leftarrow$ f1d($GK_{MTC}$, $NONCE$, $CH_{MTC}$, $GID$, $SN_{ID}$, $PATH$) ;

if $XMAC_{GK_{MTC}}$ do not equals $MAC_{GK_{MTC}}$ then
    Abandon Function;
end

$RES_D \leftarrow$ f2($GK_{MTC}$, $CH_{MTC}$);

$CK \leftarrow$ f3($GK_{MTC}$, $CH_{MTC}$);

$IK \leftarrow$ f4($GK_{MTC}$, $CH_{MTC}$);
```

6.4.6 MME and HSS: Traverse Tree

In order for the MME and HSS to find a node in the inverted hash trees we introduce a general purpose function named Traverse Tree.

Input:	(PATH, TREE HEIGHT, $Node_{root}$)
Output:	($Node_{path}$)

The function reads TREE HEIGHT number of bits from PATH and for each bit SHA-256 is performed. Depending on the bit that is read from PATH either *h0* or *h1* is performed, as defined in Section 6.4.2. If the bit is 0 then a byte of zeros is appended to the input node value, otherwise a byte of ones is appended to the input node value. This function must be standardized since it must be performed the same way by MME and HSS of different operators. Pseudo-code for this function is provided in pseudo-code 2.

6.4.7 ME: Generate NONCE

We are not aware of any pseudo-random generator function defined for the USIM application; therefore, we introduce such a function to the ME instead. It is required by every MTC device that want to be a part of the group-based AKA.

Pseudo-code 2: MME and HSS: Traverse Tree

Function for computing the node $\mathsf{Node}_{\mathsf{path}}$ *at position PATH in an inverted hash tree*

```
input  : $\mathsf{Node}_{\mathsf{root}}$, PATH, TREE HEIGHT
output : $\mathsf{Node}_{\mathsf{path}}$

Digest ← $\mathsf{Node}_{\mathsf{root}}$;

for $i$ ← 0 to TREE HEIGHT do
    current Bit ← bit $i$ of PATH;
    if current Bit equals 0 then
        Append byte of zeros to Digest;
    else
        Append byte of ones to Digest;
    end
    Digest ← SHA256(Digest);
    Digest ← truncate to 128 bits(Digest);
end
$\mathsf{Node}_{\mathsf{path}}$ ← Digest;
```

6.4.8 MME: Get GK_{MTC} and CH_{MTC}

The MME must be able to compute the CH_{MTC} and GK_{MTC} parameters by using the information received from Authentication Information Request and Attach Request; therefore, we introduce a function with the following signature.

Input:	(PATH, TREE HEIGHT, NODE DEPTH, CH_{ij}, GK_{ij})
Output:	(CH_{MTC}, GK_{MTC})

The function can be realized with the following algorithm. First the corrected path is computed. The corrected path is then used to traverse the inverted hash trees with roots CH_{ij} and GK_{ij} using the function in Section 6.4.6. Finally, the function returns CH_{MTC} and GK_{MTC}.

Pseudo-code 3: MME: Get GK_{MTC} and CH_{MTC}

```
Pseudo-code for computing the leaf values $GK_{MTC}$ and $CH_{MTC}$
input  : PATH, TREE HEIGHT, NODE DEPTH, $GK_{ij}$, $CH_{ij}$
output : $GK_{MTC}$, $CH_{MTC}$

Corr. Path Pos $\leftarrow$ (TREE HEIGHT $-$ NODE DEPTH);
Corr. PATH $\leftarrow$ bit number Corr. Path Pos to TREE HEIGHT in PATH

$GK_{MTC}$ $\leftarrow$ Traverse Tree(Corr. PATH, NODE DEPTH, $GK_{ij}$);
$CH_{MTC}$ $\leftarrow$ Traverse Tree(Corr. PATH, NODE DEPTH, $CH_{ij}$);
```

6.4.9 HSS: Generate group-authentication parameters

When the HSS receives the Authentication Information Request, it must produce CH_{ij}, GK_{ij}, IMSI, TREE HEIGHT and NODE DEPTH in order to reply. We introduce the following function to the HSS in order to achieve this.

Input:	(GID, PATH)
Output	(CH_{ij}, GK_{ij}, IMSI, TREE HEIGHT, NODE DEPTH)

The function first identifies the IMSI coupled to the GID and PATH. The function also identifies the inverted hash trees tied to the GID and retrieves the TREE HEIGHT coupled to them. Depending on policy choices made by the operator of the HSS, the function chooses a certain NODE DEPTH and traverses the inverted hash trees from the root to the these sub-tree roots using the function defined in Section 6.4.6. The function then returns CH_{ij}, GK_{ij}, IMSI, TREE HEIGHT and NODE DEPTH.

Pseudo-code 4: HSS: Generate group-authentication parameters

```
Pseudo-code for HSS retrieval of $GK_{ij}$ and $CH_{ij}$ and others
input  : GID, PATH
output : $GK_{ij}$, $CH_{ij}$, IMSI, TREE HEIGHT, NODE DEPTH

IMSI $\leftarrow$ Database look-up(GID, PATH);
Policy $\leftarrow$ Database look-up(GID, PATH);
NODE DEPTH $\leftarrow$ Database look-up(GID, PATH, Policy);

$GK_{root}$ $\leftarrow$ Database look-up(GID);
$CH_{root}$ $\leftarrow$ Database look-up(GID);
TREE HEIGHT $\leftarrow$ Database look-up(GID);

Corr. PATH $\leftarrow$ bit number 0 to NODE DEPTH in PATH
$GK_{ij}$ $\leftarrow$ Traverse Tree(Corr. PATH, NODE DEPTH, $GK_{root}$);
$CH_{ij}$ $\leftarrow$ Traverse Tree(Corr. PATH, NODE DEPTH, $CH_{root}$);
```

6.5 Storage of new structures

6.5.1 MME and HHS

How to specifically store the new structures and parameters in the MME and HSS is out of scope for this thesis. However, since these entities can be assumed to have high memory capacity we can assume the storage itself is not an issue and can therefore be left up to the implementer. We have introduced the following parameters to be supported by the MME: RES_D, GID, PATH, NONCE, TREE HEIGHT, NODE DEPTH, GK_{ij} and CH_{ij}. Futhermore, we have introduced the following parameters to be supported by the HSS: GID, PATH, NONCE, GK_{ij}, CH_{ij}, NODE DEPTH and TREE HEIGHT.

6.5.2 USIM

How to specifically store the new structures and parameters in the USIM is out of scope for this thesis. However, we think that certain elementary files in the UICC could be reused for storage of the new parameters. In particular, the elementary files GID1 and GID2 are specified to be used "to identify a group of USIMs for a particular application" as stated in clause 4.2.10 in [25]. These could be used to store the parameters PATH and GID. In this thesis, we introduce the following parameters to be supported by the USIM and the UICC: GID, PATH, NONCE, RES_D, CH_{MTC}, GK_{MTC} and O_{MTC}.

6.6 OAI reference implementation

In this section we showcase how our implementation of the aforementioned parameters, functions and messages perform in OAI by presenting some signalling examples. Moreover, the configuration of our OAI setup and the scenarios which were considered are also presented. The traverse-tree function, f1-derivative function and group-authenticate function reference implementation can be found in Appendix C. A patch file for our reference implementation in OAI can be found at `https://bitbucket.org/Thremore/group-aka`.

6.6.1 Configuration

OAI was run on three virtual machines (VMs) inside a single host computer. The host was running Linux and had an Intel Core i7 processor, 4GB RAM and used the operating system UBUNTU LTS 14.04. An overview of the setup used in this thesis can be seen in Figure 6.2.

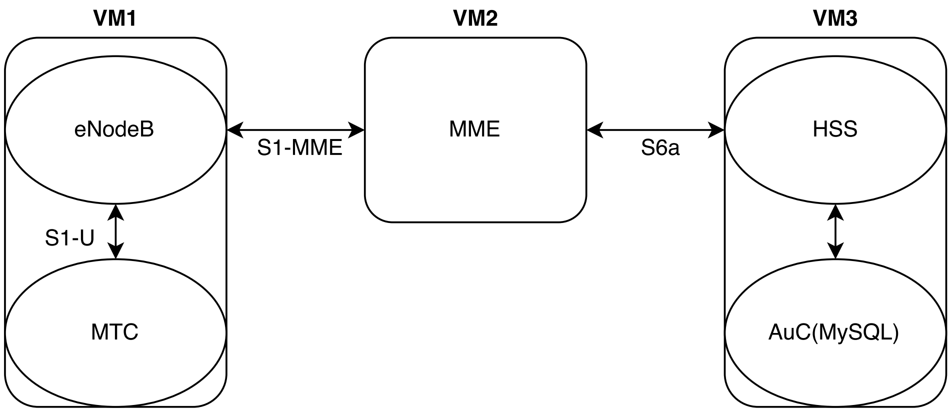


Figure 6.2: Network set-up for OAI.

The communication between the MTC device, the MME, the HSS and the eNodeB are performed through Ethernet interfaces. The channel between VM1 and VM2 represent the S1-MME interface in EPS, which supports NAS signalling. The communication between the MTC device and the eNodeB is done through VM1 and represents the S1-U interface. VM3 was running the HSS which uses a MySQL server for storage of subscriber data.

We retrieved the signalling information by using the packet analyser tool Wireshark. In accordance with the scope of the thesis, the only messages of the protocols which are analysed are between the Attach Request message and the point when K_{ASME} has been generated in both the MTC device and the MME.

6.6.2 Scenarios

We performed tests of the prototype implementation based on three different scenarios. In the first scenario an MTC device authenticates and derives keys with a serving network MME and an HSS by performing the classic EPS AKA.

In the second scenario an MTC device authenticates and derives keys with a serving network MME and an HSS by performing Case A of the new protocol. The MTC device is part a group and has been given GID and PATH parameters to which the HSS holds the corresponding authentication information.

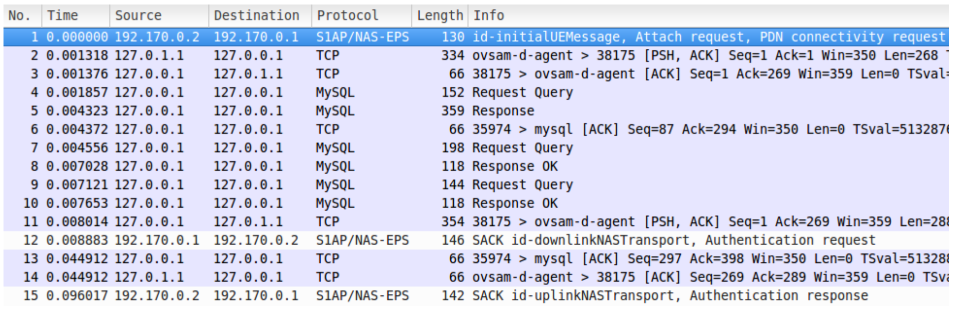
In the third scenario an MTC device authenticates and derives keys with a serving network MME and an HSS by performing Case B of the new protocol. The MTC device is part of the same group as in the Case A scenario. The MME has been pre-loaded with the required information about the group that was established during the scenario running Case A.

6.6.3 Signalling example standard EPS AKA

Performing standard EPS AKA in OAI produces the signalling depicted in Figure 6.3, as captured by Wireshark. Messages number 2 to 11 and 13 and 14 in the figure represent MME and HSS Diameter signalling and MySQL queries from the HSS. Messages 1, 12 and 15 represent the NAS signalling performed between UE and MME.

6.6.4 Signalling example group-based AKA Case A

Performing Case A of the new protocol implementation in OAI produces the signalling depicted in Figure 6.4, as captured by Wireshark. Messages 2 and 11 represent the modified Authentication Information Request and Authentication Information Answer messages between MME and HSS.

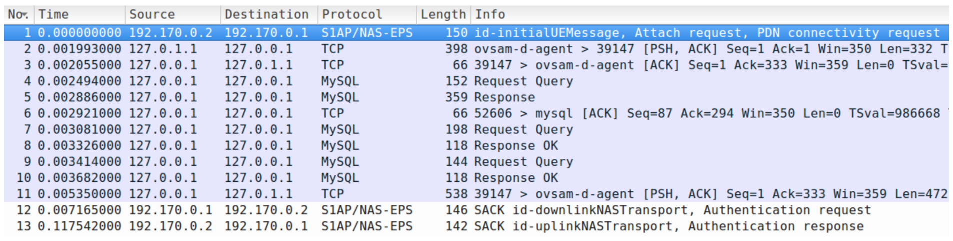


```
No. Time     Source      Destination Protocol     Length Info
  1 0.000000 192.170.0.2 192.170.0.1 S1AP/NAS-EPS    130 id-initialUEMessage, Attach request, PDN connectivity request
  2 0.001318 127.0.1.1   127.0.0.1   TCP             334 ovsam-d-agent > 38175 [PSH, ACK] Seq=1 Ack=1 Win=350 Len=268
  3 0.001376 127.0.0.1   127.0.1.1   TCP              66 38175 > ovsam-d-agent [ACK] Seq=1 Ack=269 Win=359 Len=0 TSval=
  4 0.001857 127.0.0.1   127.0.0.1   MySQL           152 Request Query
  5 0.004323 127.0.0.1   127.0.0.1   MySQL           359 Response
  6 0.004372 127.0.0.1   127.0.0.1   TCP              66 35974 > mysql [ACK] Seq=87 Ack=294 Win=350 Len=0 TSval=513287
  7 0.004556 127.0.0.1   127.0.0.1   MySQL           198 Request Query
  8 0.007028 127.0.0.1   127.0.0.1   MySQL           118 Response OK
  9 0.007121 127.0.0.1   127.0.0.1   MySQL           144 Request Query
 10 0.007653 127.0.0.1   127.0.0.1   MySQL           118 Response OK
 11 0.008014 127.0.0.1   127.0.1.1   TCP             354 38175 > ovsam-d-agent [PSH, ACK] Seq=1 Ack=269 Win=359 Len=28
 12 0.008883 192.170.0.1 192.170.0.2 S1AP/NAS-EPS    146 SACK id-downlinkNASTransport, Authentication request
 13 0.044912 127.0.0.1   127.0.0.1   TCP              66 35974 > mysql [ACK] Seq=297 Ack=398 Win=350 Len=0 TSval=51328
 14 0.044912 127.0.1.1   127.0.0.1   TCP              66 ovsam-d-agent > 38175 [ACK] Seq=269 Ack=289 Win=359 Len=0 TSv
 15 0.096017 192.170.0.2 192.170.0.1 S1AP/NAS-EPS    142 SACK id-uplinkNASTransport, Authentication response
```

Figure 6.3: Wireshark output when running standard EPS AKA.

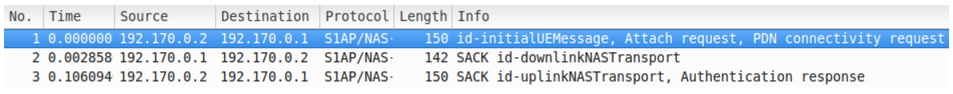
6.6.5 Signalling example group-based AKA Case B

Performing Case B of the new protocol implementation in OAI produces the signalling depicted in Figure 6.4, as captured by Wireshark. Message number 1 is the same modified Attach Request message as in Case A. The second message is the new message introduced to EPS, namely the Authentication Request Derivable. Note that Diameter signalling is not performed in this scenario.



```
No. Time        Source      Destination Protocol     Length Info
  1 0.000000000 192.170.0.2 192.170.0.1 S1AP/NAS-EPS    150 id-initialUEMessage, Attach request, PDN connectivity request
  2 0.001993000 127.0.1.1   127.0.0.1   TCP             398 ovsam-d-agent > 39147 [PSH, ACK] Seq=1 Ack=1 Win=350 Len=332 T
  3 0.002055000 127.0.0.1   127.0.1.1   TCP              66 39147 > ovsam-d-agent [ACK] Seq=1 Ack=333 Win=359 Len=0 TSval=
  4 0.002494000 127.0.0.1   127.0.0.1   MySQL           152 Request Query
  5 0.002886000 127.0.0.1   127.0.0.1   MySQL           359 Response
  6 0.002921000 127.0.0.1   127.0.0.1   TCP              66 52606 > mysql [ACK] Seq=87 Ack=294 Win=350 Len=0 TSval=986668
  7 0.003081000 127.0.0.1   127.0.0.1   MySQL           198 Request Query
  8 0.003326000 127.0.0.1   127.0.0.1   MySQL           118 Response OK
  9 0.003414000 127.0.0.1   127.0.0.1   MySQL           144 Request Query
 10 0.003682000 127.0.0.1   127.0.0.1   MySQL           118 Response OK
 11 0.005350000 127.0.0.1   127.0.1.1   TCP             538 39147 > ovsam-d-agent [PSH, ACK] Seq=1 Ack=333 Win=359 Len=472
 12 0.007165000 192.170.0.1 192.170.0.2 S1AP/NAS-EPS    146 SACK id-downlinkNASTransport, Authentication request
 13 0.117542000 192.170.0.2 192.170.0.1 S1AP/NAS-EPS    142 SACK id-uplinkNASTransport, Authentication response
```

Figure 6.4: Wireshark output when running Case A.



```
No. Time     Source      Destination Protocol Length Info
  1 0.000000 192.170.0.2 192.170.0.1 S1AP/NAS·   150 id-initialUEMessage, Attach request, PDN connectivity request
  2 0.002858 192.170.0.1 192.170.0.2 S1AP/NAS·   142 SACK id-downlinkNASTransport
  3 0.106094 192.170.0.2 192.170.0.1 S1AP/NAS·   150 SACK id-uplinkNASTransport, Authentication response
```

Figure 6.5: Wireshark output when running Case B.

Chapter 7

Performance analysis

In this chapter we analyse the performance of the new group-based Authentication and Key Agreement (AKA) protocol. The analysis is based on our prototype implementation and OpenAirInterface (OAI) configuration, which we presented in the previous chapter. The analysis is divided into two different parts: bandwidth consumption and latency.

7.1 Bandwidth consumption

In this section the bandwidth consumption in the Non-Access Stratum (NAS) and in the S6a interface for the group-based AKA is analysed.

7.1.1 Assumptions and analysis approach

Only the AKA parameters discussed in the last chapter are considered in the analysis. Since some parameters have variable sizes, for the analysis we assume some reasonable example sizes, which are given and motivated later in this subsection. The same example sizes are assumed throughout in the bandwidth consumption analysis. Two steps are taken for the analysis. The first is to separate the three cases EPS AKA, Case A and Case B and count their bandwidth consumption, for NAS communication on the one hand and for S6a communication on the other. The next step is to use the results of the first step to calculate (for NAS and S6a respectively) the total bandwidth consumption (for EPS AKA on the one hand and the group-based AKA on the other) when some different numbers of MTC devices want to connect.

Not only the actual parameters are counted in the analysis; we also include the information element headers and the attribute-value pair headers.

Two further assumptions are made in the second step for the group-based AKA case: 1) All the devices belong to the same group. 2) The devices are located in the tree such that only one Case A AKA have to be performed.

For the AKA parameters that is used in EPS AKA we assume that the sizes implied by the MILENAGE algorithm set is used.

We set the GID to 15 digits since this would allow, if the MSIN is 9 digits, to have 1 000 000 groups for the operator. It seems reasonable that an operator would

Table 7.1: Sizes of AKA parameter IEs for EPS AKA.

Parameter	**Information Element**	**Size range**	**Example size**
IMSI	EPS mobile identity	5–9 [14]	9
RAND	Authentication parameter RAND	16 [22]	16
AUTN	Authentication parameter AUTN	17 [22]	17
Authentication response parameter	Authentication response parameter	9 [2]	9

like to use that many digits for the MSIN, since this is the usual amount of digits used for IMSI MSINs, if the number of digits used for the MNC is three.

For PATH and its auxiliary parameters TREE HEIGHT and NODE DEPTH we cannot find example sizes that are obviously the best ones. Nevertheless, we have to decide sizes for the values. We let TREE HEIGHT be a function of the number of MTC devices according to the following rule. TREE HEIGHT equals 2 multiplied with the ceiling of the two logarithm of the number of MTC devices. We set the size of PATH to be the smallest number that is at least as large as TREE HEIGHT and at the same time divisible by eight. We let NODE DEPTH have the size of TREE HEIGHT divided by two. One thing can be said in defence of our assumptions: TREE HEIGHT and NODE DEPTH are proportional to the logarithm of the number of MTC devices.

We let the IMSI consist of 15 digits since 15 is the typical number of digits used.

7.1.2 Non-Access Stratum (NAS)

In the calculation of the first step of the NAS bandwidth consumption, we include bytes used for giving the types and sizes of the Information Elements (IEs) to the degree that they are specified to be included in the corresponding messages. Tables 7.1, 7.2 and 7.3 show the sizes of the IEs for EPS AKA, Case A and Case B, respectively.

The result of the second step is shown in Table 7.4.

7.1.3 S6a interface bandwidth consumption

We derive the bandwidth consumption comparison in the S6a interface by counting the bytes of the sent AKA/group-AKA parameters in the authentication information request (AIR) message and the authentication information answer (AIA) message. The AIA and AIR messages are based on the Diameter protocol, which means that each parameter is encapsulated in an AVP as described in Section 2.3.2. Every AVP value must be aligned with 32 bits and also include a header that varies

Table 7.2: Sizes of AKA parameter IEs for the group-based AKA, Case A. The values are given in bytes.

Parameter	**Information Element**	**Size range**	**Example size**
GID	EPS mobile identity	5–9	9
PATH	PATH IE	2–33	Depends
NONCE	NONCE IE	16	16
RAND	Authentication parameter RAND	16 [22]	16
AUTN	Authentication parameter AUTN	17 [22]	17
Authentication response parameter	Authentication response parameter	9 [2]	9

Table 7.3: Sizes of AKA parameter IEs for the group-based AKA, Case B. The values are given in bytes.

Parameter	**Information Element**	**Size range**	**Example size**
GID	EPS mobile identity	5–9	9
PATH	PATH IE	2–33	Depends
NONCE	NONCE IE	16	16
CH_{MTC}	CH_{MTC} IE	16	16
AUT_{D}	AUT_{D} IE	15	15
RES_{D}	Authentication response parameter	9	9

Table 7.4: NAS bandwidth consumptions for EPS AKA and the group-based AKA. The values are given in bytes.

# of MTC devices	**PATH IE size**	**EPS AKA**	**Group-based AKA**
1	2	51	69
100	3	5,100	6,802
1,000	4	51,000	69,002
10,000	5	510,000	700,002
100,000	6	5,100,000	7,100,002
1,000,000	6	51,000,000	71,000,002

Table 7.5: Parameter sizes in Diameter for EPS AKA

Parameter	AVP type	Size (bytes)
IMSI	User-Name	16
RAND	OctetString	28
XRES	OctetString	20
AUTN	OctetString	28
K_{ASME}	OctetString	44
SNid	OctetString	16

Table 7.6: Parameter sizes in Diameter for Case A

Parameter	AVP type	Size (bytes)
IMSI/GID	User-Name	16
RAND	OctetString	28
XRES	OctetString	20
AUTN	OctetString	28
K_{ASME}	OctetString	44
PATH	OctetString	16 to 20
GK	OctetString	28
CH	OctetString	28
TREE HEIGHT	Unsigned32	16
NODE DEPTH	Unsigned32	16
SN_{ID}	OctetString	16

between 8 and 12 bytes depending on an extra header structure that sometimes occur [24]. These rules give us the EPS parameter sizes in S6a shown in Table 7.5 and the group-based AKA parameter sizes in S6a shown in Table 7.6.

We extrapolate this data in order to construct Table 7.7 which presents the bandwidth consumption for when 1 up to 1,000,000 MTC devices attach in the S6a interface for EPS AKA and group-based AKA. Note that the PATH parameter grows in size with larger groups of MTC devices, so for 100,000 devices and over it is 20 bytes in size rather than 16 bytes.

7.1.4 Conclusion

In this section we present three graphs based on the bandwidth consumption functions 7.1, 7.2, 7.3 and 7.4. The functions are derived from the aforementioned parameter sizes and the EPS AKA and group-based AKA protocols. In these functions we denote by n the number of MTC devices in a group. Note that the size of the PATH parameter varies according to the size of the MTC group. It must also be transmitted on full bytes in NAS and on 32-bits in the S6a interface, which gives the expressions for *PATHsize* seen in the functions.

Table 7.7: Bandwidth consumption for EPS AKA and the group-based AKA on the S6a interface.

# of MTC devices	EPS AKA (bytes)	Group-based AKA (bytes)
1	152	304
100	15,200	304
1,000	152,000	304
10,000	1,520,000	304
100,000	15,200,000	312
1,000,000	152,000,000	312

EPS AKA NAS bandwidth consumption:

$$Bandwidth = n * (IMSI + RAND + AUTN + RES) \tag{7.1}$$

Group-based AKA NAS bandwidth consumption:

$$\begin{gathered} Bandwidth = n * (gid + PATHsize + NONCE) + \\ (n-1) * (CH + AUT_{\mathrm{D}} + RES_{\mathrm{D}}) + \\ RAND + AUTN + RES \end{gathered}$$

Where

$$PATHsize = (\lceil \log_2 n \rceil * 2 - 1) \backslash 8 + 2 \tag{7.2}$$

EPS AKA S6a interface bandwidth consumption:

$$Bandwidth = n * (IMSI + RAND + XRES + AUTN + K_{\mathrm{ASME}} + SN_{\mathrm{ID}}) \tag{7.3}$$

Group-based AKA S6a interface bandwidth consumption:

$$\begin{gathered} Bandwidth = IMSI + 2 * GID + RAND + \\ XRES + AUTN + K_{\mathrm{ASME}} + GK + CH + \\ TREE\ HEIGHT + NODE\ DEPTH + SN_{\mathrm{ID}} + PATHsize \end{gathered}$$

Where

$$PATHsize = 2 * (min(PATH) + (\lceil \log_2 n \rceil * 2 - 1)\backslash 32) * 4) \tag{7.4}$$

A graph of the comparison of NAS bandwidth consumption of EPS AKA and the group-based AKA can be seen in Figure 7.3. It illustrates a steady increase of NAS bandwidth usage when using the group-based AKA instead of EPS AKA. Figure 7.2 illustrates the EPS AKA and group-based AKA bandwidth consumption on the S6a interface. It shows that by using the group-based AKA instead of classic EPS AKA a significant amount of bytes can be saved as the MTC group size grows larger. In fact, already when more than two devices are in the same group, bandwidth is decreased on the S6a interface when using the group-based AKA instead of EPS AKA.

In Figure 7.3 it is illustrated how the increase of the NAS bandwidth fares against the decrease of S6a interface bandwidth when using group-based AKA instead of classic EPS AKA. In fact, already at a group size of three MTC devices, less bandwidth is used by the group-based AKA in total compared to EPS AKA.

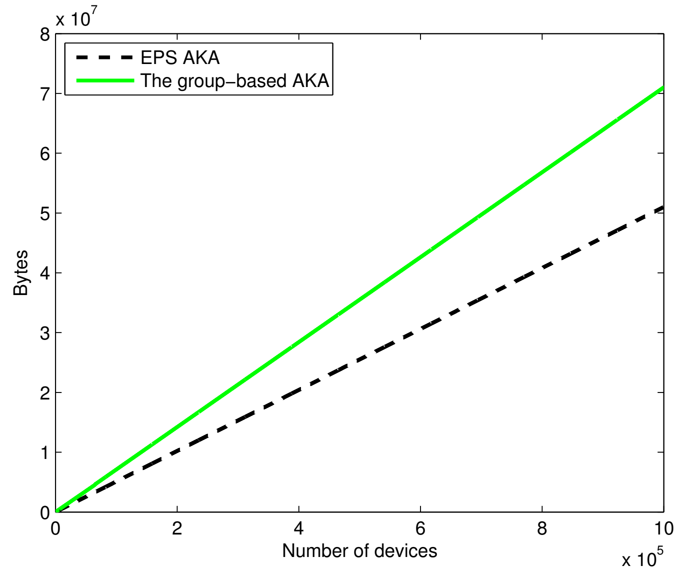


Figure 7.1: Bandwidth consumption comparison between EPS AKA and the group-based AKA in the NAS.

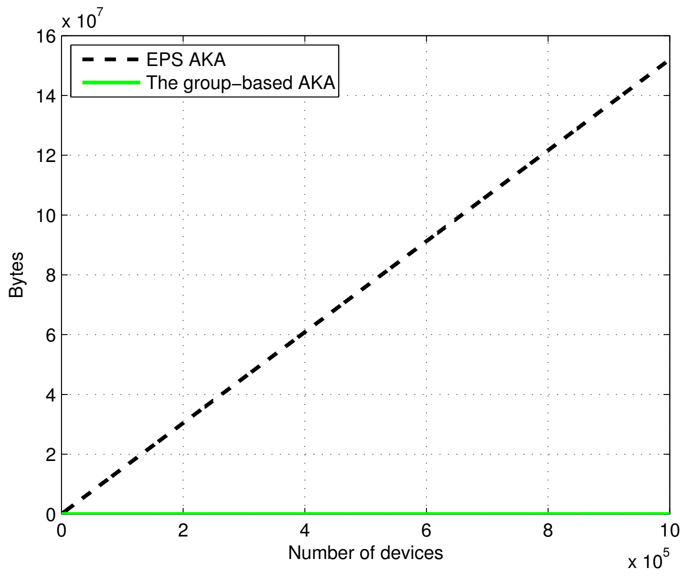


Figure 7.2: Bandwidth consumption comparison between EPS AKA and group-based AKA in the S6a interface.

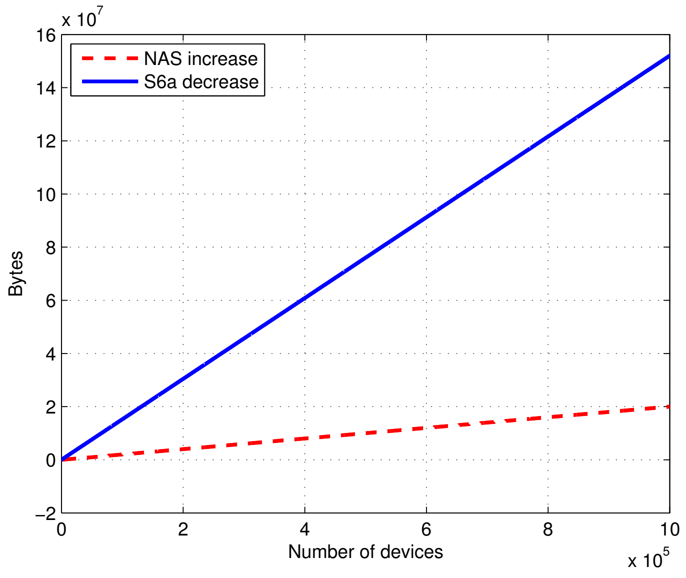


Figure 7.3: Increase in NAS bandwidth consumption and decrease in S6a bandwidth consumption when the group-based AKA is used instead of EPS AKA.

Table 7.8: Ping results

City, Country	Distance from MME (km)	RTT (ms)
Lund, Sweden	1	1
Valencia, Spain	2066	65
Cape Town, South Africa	9989	188

7.2 Latency

One of the goals of the new protocol is to reduce traffic and latency between the MME and the HSS when a massive number of MTC devices attach to the network. In order to advance this goal with some empirical data, here follows a study of how the implementation of the new protocol in OAI compared to the EPS AKA implementation with regards to latency. The latency in this study is measured as the time in milliseconds from that the UE sends the Attach Request message to when the MME sends the message Security Mode Command after computing K_{ASME}/K_{asmeD}.

The study is based on a number of assumptions regarding the round-trip times (RTTs) between the MTC device, the MME and the HSS. First, we consider an HSS which has a geographically varied distance from the MME. The reason for this is to emulate the scenario of a UE attaching from a different country than where its operator, and consequently the HSS responsible for its subscriber data, is located. Moreover, since the channel of interest is between the MME and HSS, we assume that the RTT between the UE and the MME is static in this study.

In order to estimate the RTT between the MME and the other countries, the Linux software utility *Ping* was used. By using WonderProxy servers [40] which were stationed all around the world as a reference, a number of RTTs between Lund and cities in other countries were measured. The result of this testing can be seen in Table 7.8, where each RTT value is the average of 100 trials. The estimated RTTs were used as a basis for emulating latency in the S6a interface in OAI by using the Linux traffic control tool [41].

Figure 7.4 shows the result of the latency study. It depicts the average latency when running the three different procedures 20 times in OAI for each emulated HSS distance. The result showcases that when the HSS is further away from the MME, the latency rises for EPS AKA and Case A of the group-based AKA implementation. In fact, Case A of the new protocol seems to produce more latency than EPS AKA. We suspect this is because more data is communicated. However, the latency for running Case B of the group-based AKA protocol does not rise significantly with the distance of the HSS. The reason for this is, not surprisingly, that in Case B of the new

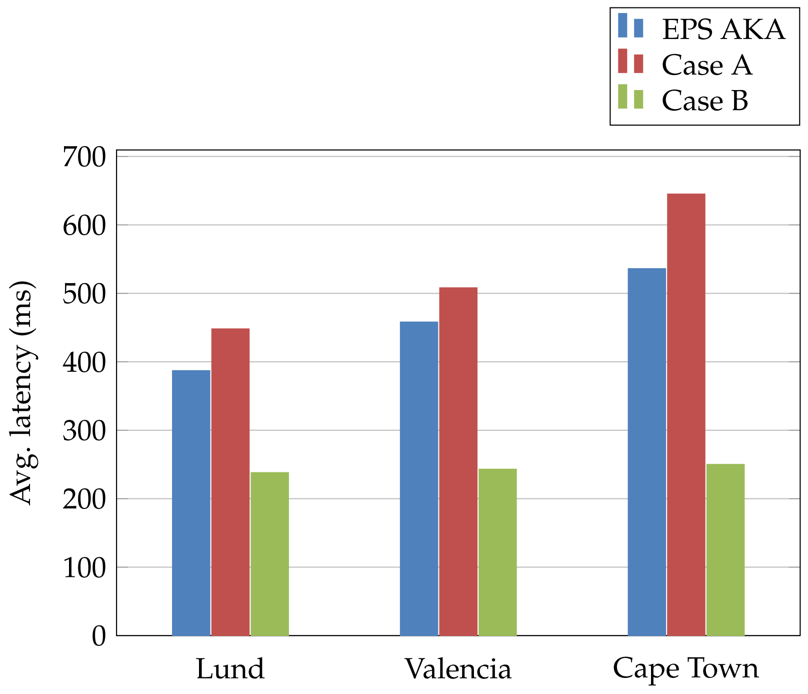


Figure 7.4: Latency comparison between procedures.

protocol the HSS is not contacted at all. From these results it follows that when an MTC device is running Case B of the new protocol, the latency is reduced.

Chapter 8

Discussion and conclusion

In this chapter we discuss topics which have affected the implementation of the new group-based AKA protocol. In particular, we discuss certain choices which have affected the implementation, the backwards compatibility of the new protocol and the general feasibility of the implementation.

8.1 GID and PATH parameters

The parameters PATH and GID were chosen to have variable sizes with the parameter TREE HEIGHT masking out the significant bits in PATH. Another solution that was discussed was using smaller and fixed sizes for these parameters. For example reusing the current IMSI structure of maximum size 15 digits to represent both the GID and the PATH. Then, after using MCC and MNC for specifying the operator's country and the operator, at least 36 bits would be left for specifying the identity of the group and giving the PATH. Simply splitting these 36 bits in half would provide each operator with 262,144 groups with equally many members for each group. Exactly the same amount of IMSIs as can be supported by each operator today could then also be placed in some group. This solution was discarded because we did not want to limit the amount of MTC devices that much. However, using fixed GID and PATH sizes could be worth considering since it might be easier to implement.

8.2 OpenAirInterface

The validity of the implementation of the new group-based AKA is limited by the fact that OpenAirInterface (OAI) is not a perfect implementation of EPS. In particular, it was found during the development of the group-based protocol implementation that OAI was lacking in areas of the EPS AKA protocol. This was because OAI was still in relatively early stages of development. For example, the handling of the IMSI, MNC and MCC values in the MME was done in a manner that did not follow 3GPP specification. This has had the affect that the internal workings of the MME while running the new group-based AKA had to be simplified regarding SN_{ID} management. However, since this simplification did not directly affect the EPS AKA signalling or authentication functions, it was ignored and should not affect the credibility of the implementation with regards to security functions and signalling accuracy.

8.3 Backwards compatibility

As shown by the implementation in this thesis, EPS networks can run the new group-based AKA with regards to signalling. However, a compatibility issue could occur when a UE that supports the implementation of the group-based AKA tries to attach to an MME which does not. Since this attach includes a GID parameter, and the MME does not support this parameter, the MME would reject any such connection. If, in the future, each MME supports the group-based AKA protocol, this becomes a non-issue. Another way to approach this issue is to always include the IMSI in the attach message and have the GID as an optional additional information element (IE). Since optional IEs do not need to be comprehended by MMEs, an MME which can not run the group-based AKA would simply ignore the GID, PATH and NONCE parameters in this solution. In fact, using this approach, such an MME would perform an EPS AKA procedure instead. However, we disregard this solution since it is too different from the protocol specification.

8.4 f1 derivative and f2–f5

Another question concerning feasibility involves the new functions in the USIM application and the MME. The new f1 derivative function that we introduced, and the functions f2–f5, which all run a specific MILENAGE version in this implementation, must be standardized in order for the group-based AKA protocol to be run. This means that all MMEs and all MTC devices that want to use the group-based AKA implementation must support these functions. This raises the question of whether all companies and organizations using the future 5G will be able to agree on this standardization or if this issue can be solved in another way.

8.5 Conclusion

In this master's thesis, state-of-the-art group-based authentication protocols have been studied and reviewed. As a result of the review it was discovered that many of the protocols were too impractical to be implemented in modern telecommunications networks. Furthermore it was also found that the proposed protocol by Choi et al. suffered from an extensive security flaw. Based on these results another group-based AKA protocol was chosen which was provided by SICS.

In the evaluation of the different EPS open-source platforms, several were found to be lacking the necessary security elements and entities needed to enable EPS AKA modification and experimentation. OAI and ns-3 were both options of interest as they both offered an extensive implementation of EPS and also supplied active communities. However, the study found that ns-3 did not provide enough security functionality to support group AKA development. We concluded that the OAI platform was best suited for implementing group-based AKA protocols and that it can provide a basis for future work in this field.

To achieve backwards compatibility and in order to provide a basis for future implementations in 5G, we provide a specification suggestion for the group-based authentication protocol. This specification reuses much of the current EPS AKA functionality. However, novel functionality and signalling has also been introduced, such as the new f1 derivative function and the new Authentication Request Derivable message.

Based on this specification, this thesis provides a reference prototype implementation of the group-based AKA protocol in OAI. Since OAI was chosen and it provides a realistic core network platform, the implementation showcases how a realistic EPS platform could be modified in order to run the new protocol. Therefore, the implementation also proves that it is in fact plausible for current mobile telecommunications systems to be modified in order to run the group-based AKA.

Lastly, a performance analysis was done which showed that the new protocol reduced traffic in the S6a interface when a lot of devices attached to the network. It was also found that the new protocol reduced latency under certain conditions in comparison with classic EPS AKA.

8.6 Future work

In order to improve the proposed group-based AKA protocol implementation, the focus should firstly be on identifying the state-machine modifications needed in the EPS AKA protocol. With the current implementation, no such considerations are taken. For example, what should happen during the event of a MTC device disconnecting or when authentication failure occurs is undefined – currently the protocol is simply terminated. Introducing these aspects to the protocol implementation could not only improve the stability of the implementation but perhaps also reveal logical issues with it. Moreover, broadening the protocol to other procedures of EPS, such as hand-over procedures and re-authentication of MTC devices, should be considered for the same reason.

Concerning performance tests, the next step to take would be to extend the OAI platform for support of a massive amount of MTC devices. During the writing of this thesis, OAI was under early development and offered little support for multiple UE and eNodeB simulations per CN. Implementing this kind of support would enable for a more realistic testing of network latency and computational effort during the scenario of a large number of MTC devices performing the EPS AKA. This could then be contrasted by similar tests using the group-based AKA, which would provide relevant research data on which to base further development.

The issue of storing the authentication parameters introduced by the group-based AKA must also be addressed. How and where such parameters should be stored in the USIM or whether its file system must be extended for this purpose requires further study.

In this thesis a UE and an MTC device have been assumed almost interchangeable. However, an MTC device cannot be assumed to have the same properties as a UE. Further study is therefore required regarding ramifications of implementing this in possibly resource-constrained MTC devices. An example study would be whether an MTC device can be expected to provide a random generator for the NONCE parameter or not.

A significant question also remains concerning the provisioning of parameters O_{MTC}, GID and PATH to the USIM application. How this could be achieved and when is something that must be specified in future implementations of this protocol. Providing a definite solution to this would further the plausibility of putting this protocol to use in 5G. There does exist certain relevant studies which have been made by 3GPP regarding over-the-air distribution of parameters to embedded UICC and trusted platforms inside MTC [42]. How the new protocol and this kind of functionality can be combined is also a question for future work.

Lastly, the implementation must also be updated since the group-based AKA protocol has been developed with additional mechanisms during this degree project.

References

[1] M. Svensson, N. Paladi, and R. Giustolisi, "5G: Towards secure ubiquitous connectivity beyond 2020." http://soda.swedishict.se/5933/, 2015. [Online; accessed 29 June 2016].

[2] 3GPP, "Universal Mobile Telecommunications System (UMTS); LTE; 3G Security; Specification of the MILENAGE algorithm set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 2: Algorithm specification," TS 35.206 V10.0.0, April 2011.

[3] 5G Infrastructure Public Private Partnership, "5G vision." https://5g-ppp.eu/wp-content/uploads/2015/02/5G-Vision-Brochure-v1.pdf, 2015. [Online; accessed 29 June 2016].

[4] NGMN Alliance, "NGMN 5G white paper." https://www.ngmn.org/fileadmin/ngmn/content/downloads/Technical/2015/NGMN_5G_White_Paper_V1_0.pdf, 2015. [Online; accessed 29 June 2016].

[5] Ericsson AB, "5G radio access." https://www.ericsson.com/res/docs/whitepapers/wp-5g.pdf, 2016. [Online; accessed 29 June 2016].

[6] Ericsson AB, "Handling of signaling storms in mobile networks." https://www.ericsson.com/res/docs/2015/handling-of-signaling-storms-in-mobile-networks-august.pdf, 2015. [Online; accessed 29 June 2016].

[7] I. Broustis, G. S. Sundaram, and H. Viswanathan, "Group authentication: A new paradigm for emerging applications," *Bell Labs Technical Journal*, vol. 17, no. 3, pp. 157–173, 2012.

[8] C. Lai, H. Li, R. Lu, and X. S. Shen, "SE-AKA: A secure and efficient group authentication and key agreement protocol for LTE networks," *Computer Networks*, vol. 57, no. 17, pp. 3492–3510, 2013.

[9] J. Cao, M. Ma, and H. Li, "GBAAM: group-based access authentication for MTC in LTE networks," *Security and Communication Networks*, vol. 8, no. 17, pp. 3282–3299, 2015.

[10] D. Choi, H.-K. Choi, and S.-Y. Lee, "A group-based security protocol for machine-type communications in LTE-advanced," *Wireless Networks*, vol. 21, no. 2, pp. 405–419, 2015.

[11] 3GPP, "LTE; General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access," TS 23.401 V10.13.0, January 2015.

[12] 3GPP, "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Vocabulary for 3GPP Specifications," TR 21.905 V10.3.0, March 2011.

[13] 3GPP, "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; 3GPP System Architecture Evolution (SAE); Security architecture," TS 33.401 V10.5.0, July 2013.

[14] 3GPP, "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); Numbering, addressing and identification," TS 23.003 V10.10.0, October 2014.

[15] 3GPP, "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Service requirements for Machine-Type Communications (MTC); Stage 1," TS 22.368 V13.1.0, March 2016.

[16] 3GPP, "LTE; Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2," TS 36.300 V10.12.0, February 2015.

[17] 3GPP, "Digital cellular telecommunications system (phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Network architecture," TS 33.102 V10.6.0, July 2014.

[18] D. Forsberg, G. Horn, W.-D. Moeller, and V. Niemi, *LTE security*. John Wiley & Sons, 2012.

[19] R. Blom, K. Norrman, M. Näslund, S. Rommer, and B. Sahlin, "Security in the Evolved Packet System," *Ericsson Review*, no. 2, pp. 4–9, 2010.

[20] 3GPP, "Universal Mobile Telecommunications System (UMTS); LTE; Cryptographic algorithm requirements," TS 33.105 V10.0.0, April 2011.

[21] 3GPP, "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Mobile radio interface signalling layer 3; General Aspects," TS 24.007 V10.0.0, March 2011.

[22] 3GPP, "Universal Mobile Telecommunications System (UMTS); LTE; Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3," TS 24.301 V10.5.0, January 2012.

[23] 3GPP, "Universal Mobile Telecommunications System (UMTS);LTE; Evolved Packet System (EPS); Mobility Management Entity (MME) and Serving GPRS Support Node (SGSN) related interfaces based on Diameter protocol," TS 29.272 V10.9.0, July 2014.

[24] V. Fajardo, J. Arkko, J. Loughney, and G. Zorn, "Diameter base protocol." RFC 6733 (Proposed Standard), Oct. 2012. Updated by RFC 7075.

[25] 3GPP, "Universal Mobile Telelecommunications System (UMTS); LTE; Characteristics of the Universal Subscriber Identity Module (USIM) application," TS 31.102 V10.12.0, January 2016.

[26] B. Blanchet, "ProVerif: Cryptographic protocol verifier in the formal model." `http://prosecco.gforge.inria.fr/personal/bblanche/proverif/`. [Online; accessed 14 August 2016].

[27] G. Piro, "LTE-Sim - the LTE simulator." `http://telematics.poliba.it/index.php/en/lte-sim`. [Online; accessed 29 June 2016].

[28] B. Wojtowicz, "OpenLTE." `https://sourceforge.net/projects/openlte`. [Online; accessed 29 June 2016].

[29] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srsLTE." `https://github.com/srsLTE/srsLTE`. [Online; accessed 29 June 2016].

[30] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srsUE." `https://github.com/srsLTE/srsue`. [Online; accessed 29 June 2016].

[31] "ns-3." https://www.nsnam.org/. [Online; accessed 12 December 2016].

[32] "LTE Module." https://www.nsnam.org/docs/models/html/lte.html. [Online; accessed 12 December 2016].

[33] "4Gsim." htts://github.com/4Gsim/4Gsim. [Online; accessed 12 December 2016].

[34] "SimuLTE-A modular system-level simulator for LTE/LTE-A networks based on OMNeT++." https://github.com/inet-framework/simulte. [Online; accessed 29 June 2016].

[35] S. Srepfler, "Python Protocol Simulator." https://sourceforge.net/p/pyprotosim/wiki/Home. [Online; accessed 29 June 2016].

[36] "OpenAirInterface." https://gitlab.eurecom.fr/oai. [Online; accessed 29 June 2016].

[37] N. Moeller, "Nettle - a low-level crypto library." https://www.lysator.liu.se/~nisse/nettle/, 2005. [Online; accessed 29 June 2016].

[38] "freeDiameter." http://www.freediameter.net/. [Online; accessed 29 June 2016].

[39] M. Bellare, J. Kilian, and P. Rogaway, "The security of the cipher block chaining message authentication code," *Journal of Computer and System Sciences*, vol. 61, no. 3, pp. 362–399, 2000.

[40] "Global proxy servers for geoip testing - WonderProxy." https://wonderproxy.com/. [Online; accessed 29 June 2016].

[41] "Traffic control." ttp://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html. [Online; accessed 29 June 2016].

[42] 3GPP, "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Feasibility study on the security aspects of remote provisioning and change of subscription for Machine to Machine (M2M) equipment," TR 33.812 V9.2.0, June 2010.

Appendix A

AKA messages

The following message structures are based on [22] and [23]. References for IEs are found in [22].

Table A.1: ATTACH REQUEST message content

IEI	Information Element	Type/Reference	Presence	Format	Length
	Protocol discriminator	Protocol discriminator 9.2	M	V	1/2
	Security header type	Security header type 9.3.1	M	V	1/2
	Attach request message identity	Message type 9.8	M	V	1
	EPS attach type	EPS attach type 9.9.3.11	M	V	1/2
	NAS key set identifier	NAS key set identifier 9.9.3.21	M	V	1/2
	EPS mobile identity	EPS mobile identity 9.9.3.12	M	LV	5-12
	UE network capability	UE network capability 9.9.3.34	M	LV	3-14
	ESM message container	ESM message container 9.9.3.15	M	LV-E	5-n
19	Old P-TMSI signature	P-TMSI signature 10.5.5.8	O	TV	4
50	Additional GUTI	EPS mobile identity 9.9.3.12	O	TLV	13
52	Last visited registered TAI	Tracking area identity 9.9.3.32	O	TV	6
5C	DRX parameter	DRX parameter 9.9.3.8	O	TV	3
31	MS network capability	MS network capability 9.9.3.20	O	TLV	4-10
13	Old location area identification	Location area identification 9.9.2.2	O	TV	6
9-	TMSI status	TMSI status 9.9.3.31	O	TV	1
11	Mobile station classmark 2	Mobile station classmark 2 9.9.2.4	O	TLV	5
20	Mobile station classmark 3	Mobile station classmark 3 9.9.2.5	O	TLV	2-34
40	Supported Codecs	Supported Codec List 9.9.2.10	O	TLV	5-n
F-	Additional update type	Additional update type 9.9.3.0B	O	TV	1
5D	Voice domain preference and UE's usage setting	Voice domain preference and UE's usage setting 9.9.3.44	O	TLV	3
D-	Device properties	Device properties 9.9.2.0A	O	TV	1
E-	Old GUTI type	GUTI type 9.9.3.45	O	TV	1
C-	MS network feature support	MS network feature support 9.9.3.20A	O	TV	1

Table A.2: AUTHENTICATION REQUEST message content

IEI	Information Element	Type/Reference	Presence	Format	Length
	Protocol discriminator	Protocol discriminator 9.2	M	V	1/2
	Security header type	Security header type 9.3.1	M	V	1/2
	Authentication request message type	Message type 9.8	M	V	1
	NAS key set identifier$_{\text{ASME}}$	NAS key set identifier 9.9.3.21	M	V	1/2
	Spare half octet	Spare half octet 9.9.2.9	M	V	1/2
	Authentication parameter RAND (EPS challenge)	Authentication parameter RAND 9.9.3.3	M	V	16
	Authentication parameter AUTN (EPS challenge)	Authentication parameter AUTN 9.9.3.2	M	LV	17

Table A.3: AUTHENTICATION RESPONSE message content

IEI	Information Element	Type/Reference	Presence	Format	Length
	Protocol discriminator	Protocol discriminator 9.2	M	V	1/2
	Security header type	Security header type 9.3.1	M	V	1/2
	Authentication response message type	Message type 9.8	M	V	1
	Authentication response parameter	Authentication response parameter 9.9.3.4	M	LV	5-17

Table A.4: Authentication information request message

```
<Authentication-Information-Request>  ::=  <Diameter Header: 318, REQ, PXY, 16777251 >
                                           <Session-Id >
                                           [ Vendor-Specific-Application-Id ]
                                           { Auth-Session-State }
                                           { Origin-Host }
                                           { Origin-Realm }
                                           [ Destination-Host ]
                                           { Destination-Realm }
                                           { User-Name }
                                           *[Supported-Features]
                                           [ Requested-EUTRAN-Authentication-Info ]
                                           [ Requested-UTRAN-GERAN-Authentication-Info ]
                                           { Visited-PLMN-Id }
                                           *[ AVP ]
                                           *[ Proxy-Info ]
                                           *[ Route-Record ]
```

Table A.5: Authentication information answer message

<Authentication-Information-Answer>	::=	<Diameter Header: 318, PXY, 16777251>
		<Session-Id >
		[Vendor-Specific-Application-Id]
		[Result-Code]
		[Experimental-Result]
		[Error-Diagnostic]
		{ Auth-Session-State }
		{ Origin-Host }
		{ Origin-Realm }
		* [Supported-Features]
		[Authentication-Info]
		*[AVP]
		*[Failed-AVP]
		*[Proxy-Info]
		*[Route-Record]

Table A.6: Requested-EUTRAN-Authentication-info

Requested- EUTRAN-Authentication-Info	::=	<AVP header: 1408 10415>
		[Number-Of-Requested-Vectors]
		[Immediate-Response-Preferred]
		[Re-synchronization-Info]
		*[AVP]

Table A.7: Authentication-Info

Authentication-Info	::=	<AVP header: 1413 10415>
		*[E-UTRAN-Vector]
		*[UTRAN-Vector]
		*[GERAN-Vector]
		*[AVP]

Table A.8: E-UTRAN-vector

E-UTRAN-Vector	::=	<AVP header: 141410415>
		[Item-Number]
		{ RAND }
		{ XRES }
		{ AUTN }
		{ K_{ASME} }
		*[AVP]

Appendix B

Perfect binary inverted hash tree

$$\begin{tikzcd}
 & & & n_{00} \arrow[dll] \arrow[drr] & & & & \\
 & n_{01} = h_0(n_{00}) \arrow[dl] \arrow[dr] & & & & n_{21=h_1(n_{00})} \arrow[dl] \arrow[dr] & & \\
n_{02} = h_0(n_{01}) \arrow[d] \arrow[dr] & & n_{12} = h_1(n_{01}) \arrow[d] \arrow[dr] & & n_{22} = h_0(n_{21}) \arrow[d] \arrow[dr] & & n_{32} = h_1(n_{21}) \arrow[d] \arrow[dr] & \\
n_{03} = h_0(n_{02}) & n_{13} = h_1(n_{02}) & n_{23} = h_0(n_{12}) & n_{33} = h_1(n_{12}) & n_{43} = h_0(n_{22}) & n_{53} = h_1(n_{22}) & n_{63} = h_0(n_{32}) & n_{73} = h_1(n_{32})
\end{tikzcd}$$

Figure B.1: An inverted hash tree of height 3

Appendix C

Prototype code

C.1 Traverse Tree function

```c


void
traverse_tree (
  uint8_t * root,
  uint8_t * path,
  uint8_t path_size,     /* how many bytes path  */
  uint32_t path_len,
  uint8_t * out)
{

  struct sha256_ctx ctx;
  uint8_t digest[32],
          rightorleft,
          currpath;

  uint8_t i, j, k;

  for (i = 0; i < 16; ++i) {
    digest[i] = root[i];
  }
  int count = 0;
  for(i = 0; i < path_size; i++ ){

    currpath = path[i];


    for(k = 0; k < (path_len > 8 ? 8 : path_len);
       k++){
      rightorleft = (currpath & ( 1 << (7 - k) )) >>
          (7 - k);

      if (rightorleft) {
        digest[16] = 0;
      } else {
        digest[16] = 255;
      }

      sha256_init(&ctx);
      sha256_update(&ctx, 17, digest);
      sha256_digest(&ctx, 32, digest);
      ++count;

    }

    if (path_len <= 8) {
      break;
    } else {
      path_len -= 8;
    }

  }

  for (i = 0; i < 16; ++i) {
    out[i] = digest[i];
  }

}
```

C.2 F1 Derivative function

```
void
f1_derivative(
  const uint8_t const gk[16],
  const uint8_t const nonce[16],
  const uint8_t const ch[16],
  const uint8_t const sn_id[3],
  const OctetString const * path,
  const groupid_t const * gid,
  const uint8_t gid_size,
  uint8_t mac_gk[8])
{
  uint8_t                               temp[16];
  uint8_t                               in1[16];
  uint8_t                               out1[16];
  uint8_t                               rijndaelInput[16];
  uint8_t                               i, j, k;
  uint16_t                              full_size;
  uint8_t                               full_input[80]; /* Max bytes that can be input is 16+16+3+32+8 */

  RijndaelKeySchedule (gk);

  /*Prepend input size into first block, max input size: (fixed) 16+16+3  +  (variable) Path 256 + GID 8 = 315 gives 2 bytes  */

  full_size = 16+16+3 + path->length + gid_size;

  full_input[0] = (uint8_t)(full_size >> 8);
  full_input[1] = (uint8_t)(full_size);

  /* fill fullinput vector with all parameters*/

  j = 2;

  for (i = 0; i < 16; i++){
    full_input[j] = nonce[i];
    j++;
  }

  for (i = 0; i < 16; i++){
    full_input[j] = ch[i];
    j++;
  }

  for (i = 0; i < gid_size; i++){
    full_input[j] = gid->u.value[i];;
    j++;
  }

  for (i = 0; i < 3; i++){
    full_input[j] = sn_id[i];
    j++;
  }

  for (i = 0; i < path->length; i++){
    full_input[j] =  path->value[i];
    j++;
  }

  k = 0;

  for(i = 0; i < 16; i++){
    rijndaelInput[i] = full_input[k];
    k++;
  }

  RijndaelEncrypt (rijndaelInput, temp);

  while (k < j) {

      for(i = 0; i < 16; i++){

          if(k == j){

              rijndaelInput[i] = (uint8_t) 128;


          } else {

              rijndaelInput[i] = full_input[k];

          }

          k++;

      }

```

```
        for (i = 0; i < 16; i++)
        rijndaelInput[i] ^= temp[i];

        RijndaelEncrypt (rijndaelInput, temp);

    }

    for (i = 0; i < 8; i++)
       mac_gk[i] = temp[i];

    return;

}
```

C.3 Group Authenticate function

```

 int usim_api_authenticate_derivable(
     const OctetString* cH_mtc_pP,
     const OctetString* autd_pP,
     OctetString* res_pP,
     OctetString* ck_pP,
     OctetString* ik_pP,
     const uint8_t const* plmn)
{
   LOG_FUNC_IN;

   int i;
   uint8_t gk[16],
           xmac[8];

   /* De-obfuscate the group key */
   f3(_usim_api_k, cH_mtc_pP->value, gk);
   for (i = 0; i < 16; ++i)
     gk[i] = gk[i] ^ _usim_obfuscated_gk[i];

   /* Compute the authentication response RES_D = f2GK
       (CH_MTC) */
   /* Compute the cipher key CK = f3GK (CH_MTC) */
   /* Compute the integrity key IK = f4GK (CH_MTC) */
```

```
  f234_opc0(gk,
  cH_mtc_pP->value ,
  res_pP->value ,
  ck_pP->value ,
  ik_pP->value);

  f1_derivative(gk,
  _emm_data.security->groupnonce.value ,
  cH_mtc_pP->value , plmn,
  &_emm_data.grouppath ,
  _emm_data.gid , xmac);

#define USIM_API_TEMP_SIZE 6
#define USIM_API_AUTD_SIZE 8

  if (memcmp(xmac,
     &autd_pP->value[USIM_API_TEMP_SIZE] ,
              USIM_API_AUTD_SIZE) != 0 ) {
    LOG_TRACE(INFO,
    "USIM-API-Comparing the XMAC with the MAC
        included in AUTD
    Failed");
  } else {
    LOG_TRACE(INFO,
    "USIM-API - Comparing the XMAC with the MAC
        included in AUTD
    Succeeded");
  }
  LOG_FUNC_RETURN (RETURNok);
}
```

Appendix D
ProVerif code

Here follows a security proof of the group-based AKA done in ProVerif and made by researchers at SICS Swedish ICT. Since an article describing the group based AKA protocol has not been published yet, the following ProVerif code is presented here to act as motivation for the security of the protocol in replacement of the future article.

The protocol was under development during the degree project and therefore an additional mechanism is included in the code which is not present in this degree project. The additional mechanism concerns a new sequence number parameter used for re-authentication. Even if the proof differs slightly from the protocol described in this degree project the main points of the protocol are still represented in the proof.

D.1 Corrupted MTCs ProVerif code

```
(* GAKA - Mutual Authentication and Secrecy
    Properties in presence of corrupted MTCs. Result:
    OK*)

free ch: channel.

type key.
type id.
type path.
type rand.
type int.
type bit.

(*Honest MTC*)
free imsi_honest: id.
```

```

const left: bit.
const right: bit.

const caseA: int.
const caseB: int.

free nas_complete_msg : bitstring.

event debugENDMTC1.
event debugENDMTC2.
event debugENDMME1.
event debugENDMME2.


event beginMTCa(id, id,id, key).
event endMTCa(id, id,id, key).
event beginMMEa(id, id,id,rand,key).
event endMMEa(id, id,id,rand,key).


event beginMTCb(path, id,id, bitstring).
event endMTCb(path, id,id, bitstring).
event beginMMEb(path, id,id,rand, key).
event endMMEb(path, id,id,rand, key).

query event (debugENDMTC1).
query event (debugENDMTC2).
query event (debugENDMME1).
query event (debugENDMME2).

(* Check authentication of MME to MTC case A *)
query x1: id, x2:id, x3: id, k: key; event
    (endMTCa(x1,x2,x3,k)) ==> event
    (beginMTCa(x1,x2,x3,k)).
(* Check authentication of MME to MTC case B *)
query x1: path, x2:id, x3:id, k: bitstring; event
    (endMTCb(x1,x2,x3,k)) ==> event
    (beginMTCb(x1,x2,x3,k)).

(* Check authentication of MTC to MME case A *)
query x1: id,  x2:id, x3:id, r: rand, k: key; event
```

```
    (endMMEa(x1,x2,x3,r,k)) ==> event
    (beginMMEa(x1,x2,x3,r,k)).
(* Check authentication of MTC to MME case B *)
query x1: path, x2:id, x3:id, r: rand, k: key; event
    (endMMEb(x1,x2,x3,r,k)) ==> event
    (beginMMEb(x1,x2,x3,r,k)).

free secret: bitstring [private].
query attacker (secret).





(* Probabilistic symmetric key encryption *)
(* Simulate communication between HSS and MME *)
fun internalsenc (bitstring,key,rand):bitstring.
reduc forall m:bitstring,k:key,r:rand;
sdec(internalsenc(m,k,r),k)=m.
letfun senc(x:bitstring,y:key)=new r:rand;
    internalsenc(x,y,r).

(* Binary Hash Tree *)
fun set_node(bitstring, bit):bitstring.

(* Hash functions *)
fun f2(bitstring):bitstring.
fun f3(bitstring):bitstring.
fun f4(bitstring):bitstring.
fun f5(bitstring):bitstring.
fun h(bitstring):bitstring.
fun hash(bitstring, bitstring):bitstring.
fun kdf(bitstring):key.
fun kdf_nas_enc(key):key.
fun kdf_nas_int(key):key.
fun nas_mac(bitstring, key): bitstring.

(* Mac *)
fun f1(bitstring, key):bitstring.

(* XOR *)
fun xor(bitstring, bitstring): bitstring.
```

```
equation forall m1: bitstring, m2: bitstring; xor(m1, xor(m1, m2)) = m2.

fun bs_to_key(bitstring): key [data, typeConverter].
fun bs_to_rand(bitstring): rand [data, typeConverter].


(* Path *)
fun get_child(path, bit): path.
reduc forall parent_path: path, pos: bit; get_parent(get_child(parent_path, pos))=parent_path.

table hss_keys(path, id, key, id, bitstring, bitstring, bitstring, bitstring).
table mme_keys(bitstring, bitstring, id, path, bitstring).


(*-----------------Protocol-------------------*)

(*--MTC--*)
let MTC (imsi_mtc: id, key_mtc: key, gid: id, path_mtc: path, sqn: bitstring, o_mtc: bitstring, pos: bit) =
new nonce_mtc: rand;
out(ch, (gid, path_mtc, nonce_mtc, pos));
in (ch, (case_x: int, aut_x: bitstring, sn_id: id, rand_x: rand));
if case_x=caseA then
   (let (xored_sqn: bitstring, mac_sn: bitstring)=aut_x in
   if sqn=xor(f5((key_mtc, rand_x)),xored_sqn) then
      (if mac_sn=f1((sqn, rand_x), key_mtc) then
   let res=f2((key_mtc, rand_x)) in
   let ck=f3((key_mtc, rand_x)) in
   let ik=f4((key_mtc, rand_x)) in
   let kasme=kdf((xored_sqn, ck, ik, sn_id)) in
         event beginMMEa (imsi_mtc, gid, sn_id, rand_x, kasme);
   out(ch, res);
   let knasenc_mtc = kdf_nas_enc(kasme) in
```

```
    let knasint_mtc = kdf_nas_int(kasme) in
    out(ch, senc(secret, knasenc_mtc));
    in (ch, (nasmsgmac: bitstring , mac_nas:
       bitstring));
    if mac_nas=nas_mac(nasmsgmac, knasint_mtc) then
    let enc_complete_msg=senc(nas_complete_msg,
       knasenc_mtc) in
    out (ch , (nas_complete_msg, enc_complete_msg,
       nas_mac(enc_complete_msg, knasint_mtc)));
    event debugENDMTC1;
          event endMTCa (imsi_mtc, gid, sn_id, kasme)
       else 0)
    else  0)
else if case_x=caseB then
    let (f5_hgkmtc_nonce: bitstring, mac_hgkmtc:
       bitstring)=aut_x in
    let hgk_mtc=xor(h((key_mtc, rand_x)),o_mtc) in
    if f5((hgk_mtc, nonce_mtc))=f5_hgkmtc_nonce then
       if mac_hgkmtc=f1((nonce_mtc, rand_x, gid,
          sn_id, path_mtc), bs_to_key(hgk_mtc)) then
          let res_b=f2((hgk_mtc, rand_x)) in
    let ck_b=f3((hgk_mtc, rand_x)) in
    let ik_b=f4((hgk_mtc, rand_x)) in
    let kasme_b=kdf((f5_hgkmtc_nonce, ck_b, ik_b,
       sn_id)) in
    event beginMMEb (path_mtc, gid, sn_id, rand_x,
       kasme_b);
    out(ch, res_b);
    let knasenc_mtc = kdf_nas_enc(kasme_b) in
    let knasint_mtc = kdf_nas_int(kasme_b) in
    out(ch, senc(secret, knasenc_mtc));
    in (ch, (nasmsgmac: bitstring , mac_nas:
       bitstring));
    if mac_nas=nas_mac(nasmsgmac, knasint_mtc) then
    let enc_complete_msg=senc(nas_complete_msg,
       knasenc_mtc) in
    out (ch , (nas_complete_msg, enc_complete_msg,
       nas_mac(enc_complete_msg, knasint_mtc)));
     event debugENDMTC2;
          event endMTCb (path_mtc, gid, sn_id,
             hgk_mtc);
    0.
```

```


(*--MME--*)
let MME_a (gid: id, path_mtc: path, sn_mme: id, hss_mme: key) =
out(ch, senc( (gid, path_mtc, sn_mme), hss_mme));
in(ch, from_hss: bitstring);
let (=gid, GKij: bitstring, CHij: bitstring, autn: bitstring, xres: bitstring, rand_hss: rand, kasme: key, imsi_mtc: id, n: bitstring, =path_mtc)=sdec(from_hss, hss_mme) in
let pathx=get_parent(path_mtc) in
insert mme_keys(GKij, CHij, gid, pathx, n);
event beginMTCa (imsi_mtc, gid, sn_mme, kasme);
out(ch, (caseA, autn, sn_mme, rand_hss));
in(ch, =xres);
let knasenc_mme = kdf_nas_enc(kasme) in
let knasint_mme = kdf_nas_int(kasme) in
new nasmsgmac: bitstring;
out(ch, (nasmsgmac, nas_mac(nasmsgmac, knasint_mme)));
in(ch, (=nas_complete_msg, enc_msg: bitstring, mac_nas: bitstring));
if mac_nas=nas_mac(enc_msg, knasint_mme) && nas_complete_msg=sdec(enc_msg, knasenc_mme) then
if imsi_mtc=imsi_honest then
out(ch, senc(secret, knasenc_mme));
event debugENDMME1;
event endMMEa (imsi_mtc, gid, sn_mme, rand_hss, kasme);
0.


let MME_b (gid: id, path_mtc: path, nonce_mtc: rand, sn_mme: id, pos: bit) =
get mme_keys(GKij, CHij, =gid, =get_parent(path_mtc), n) in
let GKmtc=set_node(GKij,pos) in
let hgkmtc=hash(GKmtc, n) in
event beginMTCb (path_mtc, gid, sn_mme, hgkmtc);
let CHmtc=set_node(CHij,pos) in
let hchmtc=hash(CHmtc, n) in
let f5_hgkmtc_nonce=f5((hgkmtc, nonce_mtc)) in
let mac_hgkmtc=f1((nonce_mtc, hchmtc, gid, sn_mme,
    path_mtc), bs_to_key(hgkmtc)) in
out(ch, (caseB, (f5_hgkmtc_nonce, mac_hgkmtc),
    sn_mme, hchmtc));
let ck=f3((hgkmtc, hchmtc)) in
let ik=f4((hgkmtc, hchmtc)) in
let kasme=kdf((f5_hgkmtc_nonce, ck, ik, sn_mme)) in
in(ch, res_d: bitstring);
if res_d=f2((hgkmtc, hchmtc)) then
let knasenc_mme = kdf_nas_enc(kasme) in
let knasint_mme = kdf_nas_int(kasme) in
new nasmsgmac: bitstring;
out(ch, (nasmsgmac, nas_mac(nasmsgmac, knasint_mme)));
in(ch, (=nas_complete_msg, enc_msg: bitstring,
    mac_nas: bitstring));
if mac_nas=nas_mac(enc_msg, knasint_mme) &&
    nas_complete_msg=sdec(enc_msg, knasenc_mme) then
(*honest MTC is left-left*)
if
    get_child(get_child(get_parent(get_parent(path_mtc)),
    left), left)=path_mtc then
out(ch, senc(secret, knasenc_mme));
event endMMEb (path_mtc, gid, sn_mme,
    bs_to_rand(hchmtc), kasme);
event debugENDMME2;
0.


let MME_init (sn_mme: id, hss_mme: key) =
in(ch, (gid: id, path_mtc: path, nonce_mtc: rand,
    =sn_mme, pos: bit));
if (path_mtc=get_child( get_parent(path_mtc), left)
    && pos=left) || (path_mtc=get_child(
    get_parent(path_mtc), right) && pos=right) then
(MME_a(gid, path_mtc, sn_mme, hss_mme) | MME_b(gid,
    path_mtc, nonce_mtc, sn_mme, pos)).


(*--HSS--*)
let HSS (sn_mme: id, mme_hss: key) =
in(ch, from_mme: bitstring);
let (gid: id, path_mtc: path, =sn_mme)=sdec(from_mme,
    mme_hss) in
get hss_keys(=path_mtc, imsi, key_mtc, =gid, sqn,
    rootG, rootR, n) in
new rand_hss: rand;
let xored_sqn=xor(f5((key_mtc, rand_hss)),sqn)    in
let mac_hss=f1((sqn, rand_hss), key_mtc) in
let xres=f2((key_mtc, rand_hss)) in
let ck=f3((key_mtc, rand_hss)) in
let ik=f4((key_mtc, rand_hss)) in
let kasme=kdf((xored_sqn, ck, ik, sn_mme)) in
let autn=(xored_sqn, mac_hss) in
out(ch, senc(  (gid, rootG, rootR, autn, xres,
    rand_hss, kasme, imsi, n, path_mtc), mme_hss)).






(*----------Main------------------*)
process

!(

new hss_mme_key: key;
new gid: id;
new sn_id: id;
new n: bitstring;
out(ch, n);

out(ch, gid);
out(ch, sn_id);


!(
HSS(sn_id, hss_mme_key)
) |


!(
MME_init(sn_id, hss_mme_key)
) |

new rootCH: bitstring;
new rootGK: bitstring;
new path_int: path;
out(ch, path_int);


!(



let pathl=get_child(path_int, left) in
out(ch, pathl);
let GKl=set_node(rootGK, left) in
let CHl=set_node(rootCH, left) in

let pathr=get_child(path_int, right) in
out(ch, pathr);
let GKr=set_node(rootGK, right) in
let CHr=set_node(rootCH, right) in

(* Honest MTC *)

let imsi_1=imsi_honest in
new mtckey_1: key;
new sqn_1: bitstring;

out(ch, imsi_1);

let path_1=get_child(pathl, left) in
let GKmtc_1=set_node(GKl, left) in
let CHmtc_1=set_node(CHl, left) in
let Hgkmtc_1=hash(GKmtc_1, n) in
let Hchmtc_1=hash(CHmtc_1, n) in

let o_1=xor(h((mtckey_1, Hchmtc_1)),Hgkmtc_1) in
insert hss_keys(path_1, imsi_1, mtckey_1, gid, sqn_1,
    GKl, CHl, n);


(* Corrupted MTCs *)

```

```
new imsi_2: id;
new mtckey_2: key;
new sqn_2: bitstring;

out(ch, imsi_2);
out(ch, mtckey_2);
out(ch, sqn_2);


let path_2=get_child(pathl, right) in
let GKmtc_2=set_node(GKl, right) in
let CHmtc_2=set_node(CHl, right) in
let Hgkmtc_2=hash(GKmtc_2, n) in
let Hchmtc_2=hash(CHmtc_2, n) in

let o_2=xor(h((mtckey_2, Hchmtc_2)),Hgkmtc_2) in
insert hss_keys(path_2, imsi_2, mtckey_2, gid, sqn_2,
    GKl, CHl, n);

out(ch, path_2);
out(ch, o_2);


new imsi_3: id;
new mtckey_3: key;
new sqn_3: bitstring;

out(ch, imsi_3);

let path_3=get_child(pathr, left) in
let GKmtc_3=set_node(GKr, left) in
let CHmtc_3=set_node(CHr, left) in
let Hgkmtc_3=hash(GKmtc_3, n) in
let Hchmtc_3=hash(CHmtc_3, n) in

let o_3=xor(h((mtckey_3, Hchmtc_3)),Hgkmtc_3) in
insert hss_keys(path_3, imsi_3, mtckey_3, gid, sqn_3,
    GKr, CHr, n);

out(ch, mtckey_3);
out(ch, sqn_3);
out(ch, path_3);
out(ch, o_3);


new imsi_4: id;
new mtckey_4: key;
new sqn_4: bitstring;

out(ch, imsi_4);

let path_4=get_child(pathr, right) in
let GKmtc_4=set_node(GKr, right) in
let CHmtc_4=set_node(CHr, right) in
let Hgkmtc_4=hash(GKmtc_4, n) in
let Hchmtc_4=hash(CHmtc_4, n) in

let o_4=xor(h((mtckey_4, Hchmtc_4)),Hgkmtc_4) in
insert hss_keys(path_4, imsi_4, mtckey_4, gid, sqn_4,
    GKr, CHr,n);

out(ch, mtckey_4);
out(ch, sqn_4);
out(ch, path_4);
out(ch, o_4);



MTC(imsi_1, mtckey_1, gid, path_1, sqn_1, o_1, left)
)




)
```

D.2 Mutual authentication properties ProVerif code

```
(* GAKA - Mutual Authentication Properties. Result:
    OK*)

free ch: channel.
```

```

type key.
type id.
type path.
type rand.
type int.
type bit.

const caseA: int.
const caseB: int.

const left: bit.
const right: bit.


free nas_complete_msg : bitstring.


event debugENDMTC1.
event debugENDMTC2.
event debugENDMME1.
event debugENDMME2.


event beginMTCa(id, id,id, key).
event endMTCa(id, id,id, key).
event beginMMEa(id, id,id,rand,key).
event endMMEa(id, id,id,rand,key).


event beginMTCb(path, id,id, bitstring).
event endMTCb(path, id,id, bitstring).
event beginMMEb(path, id,id,rand, key).
event endMMEb(path, id,id,rand, key).

query event (debugENDMTC1).
query event (debugENDMTC2).
query event (debugENDMME1).
query event (debugENDMME2).

(* Check authentication of MME to MTC case A *)
query x1: id, x2:id, x3: id, k: key; event
```

```
    (endMTCa(x1,x2,x3,k)) ==> event
    (beginMTCa(x1,x2,x3,k)).
(* Check authentication of MME to MTC case B *)
query x1: path, x2:id, x3:id, k: bitstring; event
    (endMTCb(x1,x2,x3,k)) ==> event
    (beginMTCb(x1,x2,x3,k)).

(* Check authentication of MTC to MME case A *)
query x1: id,  x2:id, x3:id, r: rand, k: key; event
    (endMMEa(x1,x2,x3,r,k)) ==> event
    (beginMMEa(x1,x2,x3,r,k)).
(* Check authentication of MTC to MME case B *)
query x1: path, x2:id, x3:id, r: rand,  k: key; event
    (endMMEb(x1,x2,x3,r,k)) ==> event
    (beginMMEb(x1,x2,x3,r,k)).

free secret: bitstring [private].
query attacker (secret).




(* Probabilistic symmetric key encryption *)
(* Simulate communication between HSS and MME *)
fun internalsenc (bitstring,key,rand):bitstring.
reduc forall m:bitstring,k:key,r:rand;
sdec(internalsenc(m,k,r),k)=m.
letfun senc(x:bitstring,y:key)=new r:rand;
    internalsenc(x,y,r).

(* Binary Hash Tree *)
fun set_node(bitstring, bit):bitstring.

(* Hash *)
fun f2(bitstring):bitstring.
fun f3(bitstring):bitstring.
fun f4(bitstring):bitstring.
fun f5(bitstring):bitstring.
fun h(bitstring):bitstring.
fun hash(bitstring, bitstring):bitstring.
fun kdf(bitstring):key.
fun kdf_nas_enc(key):key.
fun kdf_nas_int(key):key.
fun nas_mac(bitstring, key): bitstring.

(* Mac *)
fun f1(bitstring, key):bitstring.

(* XOR *)
fun xor(bitstring, bitstring): bitstring.
equation forall m1: bitstring, m2: bitstring; xor(m1,
    xor(m1, m2)) = m2.

fun bs_to_key(bitstring): key [data, typeConverter].
fun bs_to_rand(bitstring): rand [data, typeConverter].


(* Path *)
fun get_child(path, bit): path.
reduc forall parent_path: path, pos: bit;
    get_parent(get_child(parent_path,
    pos))=parent_path.

table hss_keys(path, id, key, id, bitstring,
    bitstring, bitstring, bitstring).
table mme_keys(bitstring, bitstring, id, path,
    bitstring).


(*------------------Protocol-------------------*)

(*--MTC--*)
let MTC (imsi_mtc: id, key_mtc: key, gid: id,
    path_mtc: path, sqn: bitstring, o_mtc: bitstring,
    pos: bit) =
new nonce_mtc: rand;
out(ch, (gid, path_mtc, nonce_mtc, pos));
in (ch, (case_x: int, aut_x: bitstring, sn_id: id,
    rand_x: rand));
if case_x=caseA then
    (let (xored_sqn: bitstring, mac_sn:
        bitstring)=aut_x in
    if sqn=xor(f5((key_mtc, rand_x)),xored_sqn) then
        (if mac_sn=f1((sqn, rand_x), key_mtc) then
    let res=f2((key_mtc, rand_x)) in
    let ck=f3((key_mtc, rand_x)) in
    let ik=f4((key_mtc, rand_x)) in
    let kasme=kdf((xored_sqn, ck, ik, sn_id)) in
            event beginMMEa (imsi_mtc, gid, sn_id,
                rand_x, kasme);
    out(ch, res);
    let knasenc_mtc = kdf_nas_enc(kasme) in
    let knasint_mtc = kdf_nas_int(kasme) in
    out(ch, senc(secret, knasenc_mtc));
    in (ch, (nasmsgmac: bitstring , mac_nas:
        bitstring));
    if mac_nas=nas_mac(nasmsgmac, knasint_mtc) then
    let enc_complete_msg=senc(nas_complete_msg,
        knasenc_mtc) in
    out (ch , (nas_complete_msg, enc_complete_msg,
        nas_mac(enc_complete_msg, knasint_mtc)));
    event debugENDMTC1;
            event endMTCa (imsi_mtc, gid, sn_id, kasme)
        else 0)
    else  0)
else if case_x=caseB then
    let (f5_hgkmtc_nonce: bitstring, mac_hgkmtc:
        bitstring)=aut_x in
    let hgk_mtc=xor(h((key_mtc, rand_x)),o_mtc) in
    if f5((hgk_mtc, nonce_mtc))=f5_hgkmtc_nonce then
        if mac_hgkmtc=f1((nonce_mtc, rand_x, gid,
            sn_id, path_mtc), bs_to_key(hgk_mtc)) then
            let res_b=f2((hgk_mtc, rand_x)) in
    let ck_b=f3((hgk_mtc, rand_x)) in
    let ik_b=f4((hgk_mtc, rand_x)) in
    let kasme_b=kdf((f5_hgkmtc_nonce, ck_b, ik_b,
        sn_id)) in
    event beginMMEb (path_mtc, gid, sn_id, rand_x,
        kasme_b);
    out(ch, res_b);
    let knasenc_mtc = kdf_nas_enc(kasme_b) in
    let knasint_mtc = kdf_nas_int(kasme_b) in
    out(ch, senc(secret, knasenc_mtc));
    in (ch, (nasmsgmac: bitstring , mac_nas:
        bitstring));
    if mac_nas=nas_mac(nasmsgmac, knasint_mtc) then
    let enc_complete_msg=senc(nas_complete_msg,
       knasenc_mtc) in
    out (ch , (nas_complete_msg, enc_complete_msg,
       nas_mac(enc_complete_msg, knasint_mtc)));
     event debugENDMTC2;
         event endMTCb (path_mtc, gid, sn_id,
            hgk_mtc);
    0.



(*--MME--*)
let MME_a (gid: id, path_mtc: path, sn_mme: id,
    hss_mme: key) =
out(ch, senc( (gid, path_mtc, sn_mme), hss_mme));
in(ch, from_hss: bitstring);
let (=gid, GKij: bitstring, CHij: bitstring, autn:
    bitstring, xres: bitstring, rand_hss: rand, kasme:
    key, imsi_mtc: id, n: bitstring,
    =path_mtc)=sdec(from_hss, hss_mme) in
let pathx=get_parent(path_mtc) in
insert mme_keys(GKij, CHij, gid, pathx, n);
event beginMTCa (imsi_mtc, gid, sn_mme, kasme);
out(ch, (caseA, autn, sn_mme, rand_hss));
in(ch, =xres);
let knasenc_mme = kdf_nas_enc(kasme) in
let knasint_mme = kdf_nas_int(kasme) in
out(ch, senc(secret, knasenc_mme));
new nasmsgmac: bitstring;
out(ch, (nasmsgmac, nas_mac(nasmsgmac, knasint_mme)));
in(ch, (=nas_complete_msg, enc_msg: bitstring,
    mac_nas: bitstring));
if mac_nas=nas_mac(enc_msg, knasint_mme) &&
    nas_complete_msg=sdec(enc_msg, knasenc_mme) then
out(ch, senc(secret, knasenc_mme));
event debugENDMME1;
event endMMEa (imsi_mtc, gid, sn_mme, rand_hss,
    kasme);
0.


let MME_b (gid: id, path_mtc: path, nonce_mtc: rand,
    sn_mme: id, pos: bit) =
get mme_keys(GKij, CHij, =gid, =get_parent(path_mtc),
    n) in
let GKmtc=set_node(GKij,pos) in
let hgkmtc=hash(GKmtc, n) in
event beginMTCb (path_mtc, gid, sn_mme, hgkmtc);
let CHmtc=set_node(CHij,pos) in
let hchmtc=hash(CHmtc, n) in
let f5_hgkmtc_nonce=f5((hgkmtc, nonce_mtc)) in
let mac_hgkmtc=f1((nonce_mtc, hchmtc, gid, sn_mme,
    path_mtc), bs_to_key(hgkmtc)) in
out(ch, (caseB, (f5_hgkmtc_nonce, mac_hgkmtc),
    sn_mme, hchmtc));
let ck=f3((hgkmtc, hchmtc)) in
let ik=f4((hgkmtc, hchmtc)) in
let kasme=kdf((f5_hgkmtc_nonce, ck, ik, sn_mme)) in
in(ch, res_d: bitstring);
if res_d=f2((hgkmtc, hchmtc)) then
let knasenc_mme = kdf_nas_enc(kasme) in
let knasint_mme = kdf_nas_int(kasme) in
out(ch, senc(secret, knasenc_mme));
new nasmsgmac: bitstring;
out(ch, (nasmsgmac, nas_mac(nasmsgmac, knasint_mme)));
in(ch, (=nas_complete_msg, enc_msg: bitstring,
    mac_nas: bitstring));
if mac_nas=nas_mac(enc_msg, knasint_mme) &&
    nas_complete_msg=sdec(enc_msg, knasenc_mme) then
event endMMEb (path_mtc, gid, sn_mme,
    bs_to_rand(hchmtc), kasme);
event debugENDMME2;
0.


let MME_init (sn_mme: id, hss_mme: key) =
in(ch, (gid: id, path_mtc: path, nonce_mtc: rand,
    =sn_mme, pos: bit));
if (path_mtc=get_child( get_parent(path_mtc), left)
    && pos=left) || (path_mtc=get_child(
    get_parent(path_mtc), right) && pos=right) then
(MME_a(gid, path_mtc, sn_mme, hss_mme) | MME_b(gid,
    path_mtc, nonce_mtc, sn_mme, pos)).

```

```
(*--HSS--*)
let HSS (sn_mme: id, mme_hss: key) =
in(ch, from_mme: bitstring);
let (gid: id, path_mtc: path, =sn_mme)=sdec(from_mme,
    mme_hss) in
get hss_keys(=path_mtc, imsi, key_mtc, =gid, sqn,
    rootG, rootR, n) in
new rand_hss: rand;
let xored_sqn=xor(f5((key_mtc, rand_hss)),sqn)    in
let mac_hss=f1((sqn, rand_hss), key_mtc) in
let xres=f2((key_mtc, rand_hss)) in
let ck=f3((key_mtc, rand_hss)) in
let ik=f4((key_mtc, rand_hss)) in
let kasme=kdf((xored_sqn, ck, ik, sn_mme)) in
let autn=(xored_sqn, mac_hss) in
out(ch, senc(  (gid, rootG, rootR, autn, xres,
    rand_hss, kasme, imsi, n, path_mtc), mme_hss)).






(*----------Main------------------*)
process

!(

new hss_mme_key: key;
new gid: id;
new sn_id: id;
new n: bitstring;
out(ch, n);

out(ch, gid);
out(ch, sn_id);


!(
HSS(sn_id, hss_mme_key)
) |



!(
MME_init(sn_id, hss_mme_key)
) |

new rootCH: bitstring;
new rootGK: bitstring;

new path_int: path;
out(ch, path_int);


(


let pathl=get_child(path_int, left) in
out(ch, pathl);
let GKl=set_node(rootGK, left) in
let CHl=set_node(rootCH, left) in


let pathr=get_child(path_int, right) in
out(ch, pathr);
let GKr=set_node(rootGK, right) in
let CHr=set_node(rootCH, right) in


new imsi_1: id;
new mtckey_1: key;
new sqn_1: bitstring;

out(ch, imsi_1);

let path_1=get_child(pathl, left) in
let GKmtc_1=set_node(GKl, left) in
let CHmtc_1=set_node(CHl, left) in
let Hgkmtc_1=hash(GKmtc_1, n) in
let Hchmtc_1=hash(CHmtc_1, n) in

let o_1=xor(h((mtckey_1, Hchmtc_1)),Hgkmtc_1) in
insert hss_keys(path_1, imsi_1, mtckey_1, gid, sqn_1,
    GKl, CHl, n);


(* honest MTCs *)

new imsi_2: id;
new mtckey_2: key;
new sqn_2: bitstring;

out(ch, imsi_2);

let path_2=get_child(pathl, right) in
let GKmtc_2=set_node(GKl, right) in
let CHmtc_2=set_node(CHl, right) in
let Hgkmtc_2=hash(GKmtc_2, n) in
let Hchmtc_2=hash(CHmtc_2, n) in

let o_2=xor(h((mtckey_2, Hchmtc_2)),Hgkmtc_2) in
insert hss_keys(path_2, imsi_2, mtckey_2, gid, sqn_2,
    GKl, CHl, n);




new imsi_3: id;
new mtckey_3: key;
new sqn_3: bitstring;

out(ch, imsi_3);

let path_3=get_child(pathr, left) in
let GKmtc_3=set_node(GKr, left) in
let CHmtc_3=set_node(CHr, left) in
let Hgkmtc_3=hash(GKmtc_3, n) in
let Hchmtc_3=hash(CHmtc_3, n) in

let o_3=xor(h((mtckey_3, Hchmtc_3)),Hgkmtc_3) in
insert hss_keys(path_3, imsi_3, mtckey_3, gid, sqn_3,
    GKr, CHr, n);



```

```
new imsi_4: id;
new mtckey_4: key;
new sqn_4: bitstring;

out(ch, imsi_4);

let path_4=get_child(pathr, right) in
let GKmtc_4=set_node(GKr, right) in
let CHmtc_4=set_node(CHr, right) in
let Hgkmtc_4=hash(GKmtc_4, n) in
let Hchmtc_4=hash(CHmtc_4, n) in

let o_4=xor(h((mtckey_4, Hchmtc_4)),Hgkmtc_4) in
insert hss_keys(path_4, imsi_4, mtckey_4, gid, sqn_4,
    GKr, CHr,n);



(MTC(imsi_1, mtckey_1, gid, path_1, sqn_1, o_1, left)
    | MTC(imsi_2, mtckey_2, gid, path_2, sqn_2, o_2,
    right) | MTC(imsi_3, mtckey_3, gid, path_3, sqn_3,
    o_3, left) | MTC(imsi_4, mtckey_4, gid, path_4,
    sqn_4, o_4, right)



)
)
)
```

D.3 MTC privacy related ProVerif code

```
(* GAKA - Privacy IMSI. Result: OK*)

free ch: channel.

type key.
type id.
type path.
type rand.
type int.
type bit.

free imsi_honest: id.

const caseA: int.
const caseB: int.

const left: bit.
const right: bit.

free nas_complete_msg : bitstring.

(*event debugENDMTC1.
event debugENDMTC2.
event debugENDMME1.
event debugENDMME2.



query event (debugENDMTC1).
query event (debugENDMTC2).
query event (debugENDMME1).
query event (debugENDMME2).
*)

free secret: bitstring [private].


(* Probabilistic symmetric key encryption *)
(* Simulate communication between HSS and MME *)
fun internalsenc (bitstring,key,rand):bitstring.
reduc forall m:bitstring,k:key,r:rand;
sdec(internalsenc(m,k,r),k)=m.
letfun senc(x:bitstring,y:key)=new r:rand;
    internalsenc(x,y,r).

(* Binary Hash Tree *)
fun set_node(bitstring, bit):bitstring.

(* Hash functions *)
fun f2(bitstring):bitstring.
fun f3(bitstring):bitstring.
fun f4(bitstring):bitstring.
fun f5(bitstring):bitstring.
fun h(bitstring):bitstring.
fun hash(bitstring, bitstring):bitstring.
fun kdf(bitstring):key.
fun kdf_nas_enc(key):key.
fun kdf_nas_int(key):key.
fun nas_mac(bitstring, key): bitstring.

(* Mac *)
fun f1(bitstring, key):bitstring.

(* XOR *)
fun xor(bitstring, bitstring): bitstring.
equation forall m1: bitstring, m2: bitstring; xor(m1,
    xor(m1, m2)) = m2.

fun bs_to_key(bitstring): key [data, typeConverter].
fun bs_to_rand(bitstring): rand [data, typeConverter].


(* Path *)
fun get_child(path, bit): path.
reduc forall parent_path: path, pos: bit;
    get_parent(get_child(parent_path,
    pos))=parent_path.

table hss_keys(path, id, key, id, bitstring,
    bitstring, bitstring, bitstring).
table mme_keys(bitstring, bitstring, id, path,
    bitstring).


(*------------------Protocol--------------------*)

(*--MTC--*)
let MTC (imsi_mtc: id, key_mtc: key, gid: id,
    path_mtc: path, sqn: bitstring, o_mtc: bitstring,
    pos: bit) =
new nonce_mtc: rand;
out(ch, (gid, path_mtc, nonce_mtc, pos));
```

```
in (ch, (case_x: int, aut_x: bitstring, sn_id: id, rand_x: rand));
if case_x=caseA then
    (let (xored_sqn: bitstring, mac_sn: bitstring)=aut_x in
    if sqn=xor(f5((key_mtc, rand_x)),xored_sqn) then
        (if mac_sn=f1((sqn, rand_x), key_mtc) then
    let res=f2((key_mtc, rand_x)) in
    let ck=f3((key_mtc, rand_x)) in
    let ik=f4((key_mtc, rand_x)) in
    let kasme=kdf((xored_sqn, ck, ik, sn_id)) in
    out(ch, res);
    let knasenc_mtc = kdf_nas_enc(kasme) in
    let knasint_mtc = kdf_nas_int(kasme) in
    out(ch, senc(secret, knasenc_mtc));
    in (ch, (nasmsgmac: bitstring , mac_nas: bitstring));
    if mac_nas=nas_mac(nasmsgmac, knasint_mtc) then
    let enc_complete_msg=senc(nas_complete_msg, knasenc_mtc) in
    out (ch , (nas_complete_msg, enc_complete_msg, nas_mac(enc_complete_msg, knasint_mtc)))
(*   event debugENDMTC1;*)
        else 0)
    else  0)
else if case_x=caseB then
    let (f5_hgkmtc_nonce: bitstring, mac_hgkmtc: bitstring)=aut_x in
    let hgk_mtc=xor(h((key_mtc, rand_x)),o_mtc) in
    if f5((hgk_mtc, nonce_mtc))=f5_hgkmtc_nonce then
        if mac_hgkmtc=f1((nonce_mtc, rand_x, gid, sn_id, path_mtc), bs_to_key(hgk_mtc)) then
            let res_b=f2((hgk_mtc, rand_x)) in
    let ck_b=f3((hgk_mtc, rand_x)) in
    let ik_b=f4((hgk_mtc, rand_x)) in
    let kasme_b=kdf((f5_hgkmtc_nonce, ck_b, ik_b, sn_id)) in

    out(ch, res_b);
    let knasenc_mtc = kdf_nas_enc(kasme_b) in
    let knasint_mtc = kdf_nas_int(kasme_b) in
    out(ch, senc(secret, knasenc_mtc));
    in (ch, (nasmsgmac: bitstring , mac_nas:
       bitstring));
    if mac_nas=nas_mac(nasmsgmac, knasint_mtc) then
    let enc_complete_msg=senc(nas_complete_msg,
       knasenc_mtc) in
    out (ch , (nas_complete_msg, enc_complete_msg,
       nas_mac(enc_complete_msg, knasint_mtc)));
(*     event debugENDMTC2; *)
    0.


(*−−MME−−*)
let MME_a (gid: id, path_mtc: path, sn_mme: id,
    hss_mme: key) =
out(ch, senc( (gid, path_mtc, sn_mme), hss_mme));
in(ch, from_hss: bitstring);
let (=gid, GKij: bitstring , CHij: bitstring , autn:
    bitstring , xres: bitstring , rand_hss: rand, kasme:
    key, imsi_mtc: id, n: bitstring ,
    =path_mtc)=sdec(from_hss, hss_mme) in
let pathx=get_parent(path_mtc) in
insert mme_keys(GKij, CHij, gid, pathx, n);
out(ch, (caseA, autn, sn_mme, rand_hss));
in(ch, =xres);
let knasenc_mme = kdf_nas_enc(kasme) in
let knasint_mme = kdf_nas_int(kasme) in
new nasmsgmac: bitstring;
out(ch, (nasmsgmac, nas_mac(nasmsgmac, knasint_mme)));
in(ch, (=nas_complete_msg, enc_msg: bitstring ,
    mac_nas: bitstring));
if mac_nas=nas_mac(enc_msg, knasint_mme) &&
    nas_complete_msg=sdec(enc_msg, knasenc_mme) then
out(ch, senc(secret, knasenc_mme));
(*event debugENDMME1;*)
0.


let MME_b (gid: id, path_mtc: path, nonce_mtc: rand,
    sn_mme: id, pos: bit) =
get mme_keys(GKij, CHij, =gid, =get_parent(path_mtc),
    n) in
let GKmtc=set_node(GKij,pos) in
let hgkmtc=hash(GKmtc, n) in
let CHmtc=set_node(CHij,pos) in
let hchmtc=hash(CHmtc, n) in
let f5_hgkmtc_nonce=f5((hgkmtc, nonce_mtc)) in
let mac_hgkmtc=f1((nonce_mtc, hchmtc, gid, sn_mme,
    path_mtc), bs_to_key(hgkmtc)) in
out(ch, (caseB, (f5_hgkmtc_nonce, mac_hgkmtc),
    sn_mme, hchmtc));
let ck=f3((hgkmtc, hchmtc)) in
let ik=f4((hgkmtc, hchmtc)) in
let kasme=kdf((f5_hgkmtc_nonce, ck, ik, sn_mme)) in
in(ch, res_d: bitstring);
if res_d=f2((hgkmtc, hchmtc)) then
let knasenc_mme = kdf_nas_enc(kasme) in
let knasint_mme = kdf_nas_int(kasme) in
out(ch, senc(secret, knasenc_mme));
new nasmsgmac: bitstring;
out(ch, (nasmsgmac, nas_mac(nasmsgmac, knasint_mme)));
in(ch, (=nas_complete_msg, enc_msg: bitstring,
    mac_nas: bitstring));
if mac_nas=nas_mac(enc_msg, knasint_mme) &&
    nas_complete_msg=sdec(enc_msg, knasenc_mme) then
out(ch, senc(secret, knasenc_mme));
(*event debugENDMME2;*)
0.


let MME_init (sn_mme: id, hss_mme: key) =
in(ch, (gid: id, path_mtc: path, nonce_mtc: rand,
    =sn_mme, pos: bit));
if (path_mtc=get_child( get_parent(path_mtc), left)
    && pos=left) || (path_mtc=get_child(
    get_parent(path_mtc), right) && pos=right) then
(MME_a(gid, path_mtc, sn_mme, hss_mme) | MME_b(gid,
    path_mtc, nonce_mtc, sn_mme, pos)).


(*--HSS--*)
let HSS (sn_mme: id, mme_hss: key) =
in(ch, from_mme: bitstring);
let (gid: id, path_mtc: path, =sn_mme)=sdec(from_mme,
    mme_hss) in
```

```
get hss_keys(=path_mtc, imsi, key_mtc, =gid, sqn,
    rootG, rootR, n) in
new rand_hss: rand;
let xored_sqn=xor(f5((key_mtc, rand_hss)),sqn)    in
let mac_hss=f1((sqn, rand_hss), key_mtc) in
let xres=f2((key_mtc, rand_hss)) in
let ck=f3((key_mtc, rand_hss)) in
let ik=f4((key_mtc, rand_hss)) in
let kasme=kdf((xored_sqn, ck, ik, sn_mme)) in
let autn=(xored_sqn, mac_hss) in
out(ch, senc(  (gid, rootG, rootR, autn, xres,
    rand_hss, kasme, imsi, n, path_mtc), mme_hss)).






(*------------Main--------------------*)
process

!(

new hss_mme_key: key;
new gid: id;
new sn_id: id;
new n: bitstring;
out(ch, n);

out(ch, gid);
out(ch, sn_id);


!(
HSS(sn_id, hss_mme_key)
) |


!(
MME_init(sn_id, hss_mme_key)
) |

new rootCH: bitstring;
new rootGK: bitstring;


!(


new path_int: path;
out(ch, path_int);


let pathl=get_child(path_int, left) in
out(ch, pathl);
let GKl=set_node(rootGK, left) in
let CHl=set_node(rootCH, left) in

let pathr=get_child(path_int, right) in
out(ch, pathr);
let GKr=set_node(rootGK, right) in
let CHr=set_node(rootCH, right) in

(* Honest MTC *)

let imsi_1=imsi_honest in
new mtckey_1: key;
new sqn_1: bitstring;

out(ch, imsi_1);

let path_1=get_child(pathl, left) in
let GKmtc_1=set_node(GKl, left) in
let CHmtc_1=set_node(CHl, left) in
let Hgkmtc_1=hash(GKmtc_1, n) in
let Hchmtc_1=hash(CHmtc_1, n) in

let o_1=xor(h((mtckey_1, Hchmtc_1)),Hgkmtc_1) in
insert hss_keys(path_1, imsi_1, mtckey_1, gid, sqn_1,
    GKl, CHl, n);


(* Corrupted MTCs *)

```

```
new imsi_2: id;
new mtckey_2: key;
new sqn_2: bitstring;

out(ch, imsi_2);
out(ch, mtckey_2);
out(ch, sqn_2);


let path_2=get_child(pathl, right) in
let GKmtc_2=set_node(GKl, right) in
let CHmtc_2=set_node(CHl, right) in
let Hgkmtc_2=hash(GKmtc_2, n) in
let Hchmtc_2=hash(CHmtc_2, n) in

let o_2=xor(h((mtckey_2, Hchmtc_2)),Hgkmtc_2) in
insert hss_keys(path_2, imsi_2, mtckey_2, gid, sqn_2,
    GKl, CHl, n);

out(ch, path_2);
out(ch, o_2);


new imsi_3: id;
new mtckey_3: key;
new sqn_3: bitstring;

out(ch, imsi_3);

let path_3=get_child(pathr, left) in
let GKmtc_3=set_node(GKr, left) in
let CHmtc_3=set_node(CHr, left) in
let Hgkmtc_3=hash(GKmtc_3, n) in
let Hchmtc_3=hash(CHmtc_3, n) in

let o_3=xor(h((mtckey_3, Hchmtc_3)),Hgkmtc_3) in
insert hss_keys(path_3, imsi_3, mtckey_3, gid, sqn_3,
    GKr, CHr, n);

out(ch, mtckey_3);
out(ch, sqn_3);
out(ch, path_3);
out(ch, o_3);


new imsi_4: id;
new mtckey_4: key;
new sqn_4: bitstring;

out(ch, imsi_4);

let path_4=get_child(pathr, right) in
let GKmtc_4=set_node(GKr, right) in
let CHmtc_4=set_node(CHr, right) in
let Hgkmtc_4=hash(GKmtc_4, n) in
let Hchmtc_4=hash(CHmtc_4, n) in

let o_4=xor(h((mtckey_4, Hchmtc_4)),Hgkmtc_4) in
insert hss_keys(path_4, imsi_4, mtckey_4, gid, sqn_4,
    GKr, CHr,n);

out(ch, mtckey_4);
out(ch, sqn_4);
out(ch, path_4);
out(ch, o_4);


( MTC(choice[imsi_1,imsi_2], mtckey_1, gid, path_1,
    sqn_1, o_1, left) | MTC(choice[imsi_2,imsi_1],
    mtckey_2, gid, path_2, sqn_2, o_2, right))
)

)
```