An Implementation of 5G-AKA and a Usability Analysis of OpenLDAP Access Control Lists (ACLs)

by

Rahul Punchhi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Computer Engineering

Waterloo, Ontario, Canada, 2021

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We address two pieces of work: (i) an implementation of the Authentication and Key Agreement protocol suite from the 5th generation cellular communications standards (5G-AKA) that we intend to make available as open-source, and, (ii) a categorization using Hierarchical Task Analysis (HTA) of errors made by human participants in a study carried out on the usability of Access Control Lists (ACLs) in the OpenLDAP directory.

Our work (i) on 5G-AKA is motivated by the lack of availability of such an implementation that can then be used by researchers and practitioners for further work. We discuss design choices we have made; for example, our choice of the Java programming language and cryptographic packages, and our choice to model the three entities that communicate using 5G-AKA, the User Equipment (UE), the Serving Network (SN), and the Home Network (HN) as three distinct processes that communicate over TCP sockets. We also discuss challenges we encountered in carrying out our work, and the manner in which we plan to make our work available as open-source.

Our work (ii) on error-identification in the use of ACLs in OpenLDAP is part of a broader human-subject study that, in turn, is motivated by public pronouncements of their poor usability. We discuss what HTAs are, and why they are appropriate for our work. We present our design of the HTAs, the errors we identified using them, and observe that this work helps with a prospective redesign of ACLs for OpenLDAP.

Acknowledgements

I would like to thank my mother, father, and younger brother for their love and support.

I would like to thank my supervisors, Professor Mahesh Tripunitara and Professor Catherine Rosenberg, for their guidance and support during my MASc research work.

I would like to thank Mohamed El Massad for his advice during my MASc research work.

I would like to thank Yi Fei Chen for his prior work in the design and carrying out of the OpenLDAP human-subject study.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

3GPP 3rd Generation Partnership Project 2

5G-GUTI 5G Global Unique Temporary Identifier 15

ABBA Anti-Bidding down Between Architectures 39

ACL Access Control List 53

AK Anonymity Key 34

AMF Access and Mobility Management Function 33

ARPF Authentication credential Repository and Processing Function 11

AUSF AUthentication Server Function 11

AUTN Authentication Token 34

CK Cipher Key 34, 35

DIT Directory Information Tree 53

EC Elliptic Curve 19

ECCDH Elliptic Curve Cofactor Diffie-Hellman 20

ECIES Elliptic Curve Integrated Encryption Scheme 19

eMBB Enhanced Mobile Broadband 8

EPS Evolved Packet System 7

EPS-AKA Evolved Packet System - Authentication and Key Agreement 7

ETSI-SMG European Telecommunications Standards Institute - Special Mobile Group 6

GSM Global System for Mobile Communications 6

HN Home Network 2

HSS Home Subscriber Server 7

HTA Hierarchical Task Analysis 4, 61

ICB Initial Counter Block 26

IK Integrity Key 34, 35

IMSI International Mobile Subscription Identity 17

IMT-2000 International Mobile Telecommunications 7

IoT Internet of Things 8

ITU-R International Telecommunication Union Radiocommunication Sector 8

KDF Key Derivation Function 19

LDAP Lightweight Directory Access Protocol 3

MAC Message Authentication Code 9

MCC Mobile Country Code 17

mMTC Massive Machine-type Communications 8

MNC Mobile Network Code 17

MSIN Mobile Subscriber Identification Number 17

NAS Non-Access-Stratum 15

ngKSI 5G Key Set Identifier 39

PLMN Public Land Mobile Network 17

RAND A randomly-generated value 33

RES Response value 40

SEAF SEcurity Anchor Function 11

SIDF Subscription Identifier De-concealing Function 12

SN Serving Network 2

SNPN Stand-alone Non-Public Network 17

SUCI Subscription Concealed Identifier 16

SUPI Subscription Permanent Identifier 17

TCP/IP Transmission Control Protocol/Internet Protocol 2

THEA Technique for Human Error Assessment 4, 61

UDM Unified Data Management 11

UE User Equipment 2

UICC UMTS IC Card 7

UMTS Universal Mobile Telecommunications System 7

URLLC Ultra-reliable and Low Latency Communications 8

USIM Universal Subscriber Identity Module 7

UTRA Universal Terrestrial Radio Access 7

WCDMA Wideband Code Division Multiple Access 7

XOR Exclusive-or operation 35

XRES Expected Response 34

Chapter 1

Introduction

In this chapter, we provide an introduction to two projects: (i) an implementation of the 5G-AKA protocol, and, (ii) a usability analysis of OpenLDAP ACLs using HTAs.

Our work (i) was motivated by the need for a functional proof-of-concept implementation of the 5G-AKA protocol. We sought to build a working 5G-AKA protocol that would perform each step of the authentication process in a clear and understandable way. This would mean displaying the outputs of the cryptographic functions performed at each protocol step. Displaying such values to a command-line terminal in a readable format would communicate how the 5G-AKA protocol works to users. To our knowledge, no such implementation of 5G-AKA exists that can be used by researchers and practitioners.

The motivation behind our work (ii) was to better understand the root causes of the poor usability of OpenLDAP. Prior work (cited from [25]) has already established that OpenLDAP does indeed suffer from poor usability. We sought to better understand what specific aspects of the OpenLDAP protocol may be responsible for this poor usability. We use error categorization to analyze and understand the causes of errors made by participants in the human-subject study. Our error categorization has informed our explanations for the causes of poor usability of OpenLDAP. Our error categorization has also informed our proposed ideas for how to redesign OpenLDAP to improve usability.

1.1 5G-AKA Implementation

The Authentication and Key Agreement (AKA) protocol is used in a 5G cellular network to provide mutual authentication between users and network entities. 5G is the newest generation of cellular technologies that is being standardized by the 3rd Generation Partnership Project (3GPP), a standardization body that pertains to "...cellular telecommunications technologies, including radio access, core network and service capabilities, which provide a complete system description for mobile telecommunications" [27].

Within the 5G-AKA protocol, there are 3 entities: User Equipment (UE), Serving Network (SN), and Home Network (HN). The UE represents a user device with a USIM component (e.g. smartphone) (pp. 1385) [24]. The SN represents base station network towers, which communicates with the UE via insecure wireless channels (pp. 1385) [24]. The HN represents a carrier network, which communicates with the SN via secure (authenticated) wired channels (pp. 1385) [24].

In this thesis we discuss our implementation of the 5G-AKA protocol. Our implementation is based on published 3GPP documentation. We have developed the protocol implementation in the Java programming language. Our implementation consists of three Java processes: a UE process, SN process, and HN process. These three processes correspond to the three 5G-AKA entities described previously. Our implementation uses Transmission Control Protocol/Internet Protocol (TCP/IP) [28] messages sent between Java processes to represent the authentication messages sent between network entities. As part of our implementation, we have developed functions that utilize Java cryptographic libraries (java.security and javax.crypto) and BouncyCastle API.

1.1.1 Related Work

Research pertaining to the 5G-AKA protocol was completed by David Basin et al. in a paper titled *A Formal Analysis of 5G Authentication* [24]. This paper provides a "formal model" (pp. 1383) [24] of the 5G-AKA protocol. The paper also provides "...explicit recommendations with provably secure fixes for the attacks and weaknesses..." (pp. 1383) [24] that they have identified in 5G-AKA. We follow some of these recommendations in our 5G-AKA implementation.

A test bed demonstration of the 5G-AKA protocol was proposed by Sriraam SV et al. in a paper titled *Implementation of 5G Authentication and Key Agreement Protocol on Xbee Networks* [29]. In this paper, they proposed a test bed using "...XBee S2C devices operating at 2.4GHz with a bandwidth of 5MHz supporting a data rate of upto 250 Kbps" (pp. 697) [29]. A proposed 5G test bed - to be implemented across multiple universities in India - has also been described by Rohit Budhiraja [30]. It should be noted that the paper by Sriraam SV et al. describes a "...proposed test bed demonstration..." (pp. 697) [29]. Even the multi-institute test bed project described in [30] appears to still be in development. In this thesis, we discuss our development of a completed 5G-AKA protocol implementation using Java.

1.1.2 Outline

We discuss our 5G-AKA proof-of-concept implementation in Chapter 2. In this chapter, we begin with a background on cellular network technology and security. We discuss also the development of project requirements and technical expectations for the 5G-AKA protocol implementation. In this chapter we summarize our implementation of the 5G-AKA protocol. We discuss the project design, file structuring, pre-5G-AKA initiation, and the 5G-AKA protocol implementation itself. We discuss also the protocol's error handling methods, and how this is incorporated into our implementation. We discuss the lack of user-side confirmation for successful authentication in 5G-AKA 3GPP protocol specifications, and how this has been remedied in our implementation using solutions suggested by Basin et al. [24]. Lastly, in this chapter we discuss the open-source nature of the project. We discuss also the best license to be used when the 5G-AKA implementation code is released through a public repository.

1.2 Usability Analysis of OpenLDAP ACLs

Lightweight Directory Access Protocol (LDAP) (pp. 1) [31] is designed to provide "...access to distributed directory services that act in accordance with X.500 data and service models" (pp. 1) [31]. OpenLDAP is described as an "open-source implementation" of LDAP [32]. OpenLDAP is a software protocol used to help manage access permissions within a directory. Prior work [25] [26] has already been completed to assess the usability of OpenLDAP. This prior work includes a human-subject study that trained participants

to implement an access control policy. This study evaluated these implementations and used the collected data to determine potential usability issues with OpenLDAP.

In this thesis, we discuss how we have continued this usability analysis through error categorization. We establish categories for errors made by the study's participants in their OpenLDAP policy implementations. We also develop analysis diagrams pertaining to the sample policy items used in the human-subject study. The participants' errors are analyzed using these classifications and diagrams. The specifics of this work are discussed in greater detail in Chapter 3.

1.2.1 Related Work

Prior work has been completed as part of an MASc thesis by Yi Fei Chen titled *Usability of the Access Control System for OpenLDAP* [25]. Yi Fei Chen's thesis summarizes a human-subject study where participants must "...translate a high-level policy into a corresponding OpenLDAP policy that accomplish the goals of the high-level policy" (pp. 16) [25]. Their thesis explains the structure of the study, and also summarizes the data collected. Their thesis has found that the study's participants made a considerable amount of mistakes in their policy implementations. We use the data collected from Yi Fei Chen's study in our error categorization. The contents of Chapter 3 is partially based upon this cited paper.

Prior work has also been completed as part of a research paper titled *The Poor Usability of OpenLDAP Access Control Lists (ACLs)* by Professor Mahesh Tripunitara, Yi Fei Chen, and Rahul Punchhi [26]. We summarize the error categorization process and the subsequent data analysis discussed in this prior work. The contents of Chapter 3 is also partially based upon this cited paper.

The error categorization work discussed in this thesis is completed using Hierarchical Task Analysis (HTA). HTA diagrams are used in the Technique for Human Error Assessment (THEA) approach to error categorization. These concepts are discussed in two papers. The first paper is titled *THEA: A Technique for Human Error Assessment Early in Design* by Pocock et al. [33]. The second paper is titled *THEA - A Reference Guide* by Pocock et al. [34]. We use the concept of "cognitive failure" categories outlined in [33] and [34] as a basis for our error categorization. We have also used the sample HTA diagrams

shown in (pp. 10, 23) [34] as a basis for developing our own HTA diagrams.

1.2.2 Outline

This thesis discusses the OpenLDAP error categorization in Chapter 3. This chapter begins with a background on OpenLDAP software, as well as a summary of prior work [25] and [26], including the human-subject study described in [25]. We discuss the structure of this study, the training that each participant received, and the sample policy that each participant attempted to implement. We summarize the data collected from the human-subject study. We also summarize the study's data analysis originally discussed in [26] and [25].

We discuss also the error categorization process and the diagrams that we have used in our approaches. We justify the structure of each diagram, and how it breaks down the human-subject study's sample policy items. Lastly, we analyze the results of our error categorization. This analysis includes a discussion on the prevalence of each type of error, and proposed ideas for any future redesigns to improve usability. This analysis contains content cited from [26].

Chapter 2

Implementation of 5G-AKA

In this chapter we discuss the implementation of the 5G-AKA authentication protocol using the Java programming language. We summarize the background technical history of cellular network generations, 4G network authentication, and 5G technical improvements in Section 2.1. We discuss the development of project requirements for the 5G-AKA protocol implementation in Section 2.2. We summarize all technical aspects of the 5G-AKA protocol outlined by the 3GPP in Section 2.3. This includes the individual protocol steps, necessary cryptographic functions, and message formats. In Section 2.3, we discuss also the details of our development of the 5G-AKA protocol using Java. This includes libraries used, code snippets, etc. Lastly, in Section 2.4, we discuss the open-source nature of the project, and the proposed publishing license.

2.1 Background

First generation (1G) network technology emerged during the 1980s, using "...analog technologies dedicated to provide speech services, where each call used a separated narrowband frequency channel" (pp. 1127) [35]. The 2G digital cellular network system emerged in the 1990s with the Global System for Mobile Communications (GSM) standard pushed by the European Telecommunications Standards Institute - Special Mobile Group (ETSI-SMG) (pp. 1127-1128) [35]. The services provided by 2G GSM included "...speech and circuit-switched data in macro cells..." (pp. 1128) [35]. The maximum achievable data rate was 9.6 kbps (pp. 1128) [35].

The 2G network suffered from a "lack of network authentication..." (pp. 1) [36]. Consequently, faked base stations could trick user equipment into registering with them instead of real cellular network base stations (pp. 1) [36]. These faked base stations could then deliver text messages to user equipment in order to "...attempt to defraud the user" (pp. 1) [36].

The 3G cellular network emerged as a successor to 2G in 1999 with the International Mobile Telecommunications (IMT-2000) framework specification by the ITU (pp. 1129) [35]. The 3GPP group brought forth the candidate technology of Universal Mobile Telecommunications System (UMTS) (pp. 1129) [35]. UMTS used Wideband Code Division Multiple Access (WCDMA) as the "...main air interface" (pp. 1129) [35]. This is also referred to as Universal Terrestrial Radio Access (UTRA) (pp. 1129) [35]. This allowed 3G to offer a maximum data rate of 2 Mbps (pp. 1129) [35].

The 4G cellular network system was designed to meet technical specifications which included a peak data rate of 100 Mbps to 1 Gbps (pp. 1130) [35]. The three main components of modern cellular network authentication are the UE (User Equipment), SN (Serving Network), and HN (Home Network) (pp. 2) [36]. In 4G, the UE includes a UMTS IC Card (UICC) and Universal Subscriber Identity Module (USIM) that facilitates authentication (pp. 2) [36]. The 4G SN can typically include an eNodeB base station(s) and Mobility Management Entities (pp. 2) [36]. In 4G, the HN includes the Home Subscriber Server (HSS), a component that "...stores user credentials and authenticates users" (pp. 2) [36]. The UE and SN communicate via radio access, while SN and HN communicate via IP network (pp. 2) [36]. The term Evolved Packet System (EPS) encapsulates all network entities connected via IP in the context of 4G authentication (pp. 2) [36].

The 4G network used the Evolved Packet System - Authentication and Key Agreement (EPS-AKA) authentication protocol (pp. 2) [36]. Although this authentication protocol was an improvement over 2G and 3G authentication security, it had two major weaknesses. First, the user equipment identity was sent un-encrypted (using plain text) over radio networks (pp. 3-4) [36]. Although a temporary identifier could be used to hide the UE subscriber's long-term identity, the allocation of these temporary identifiers was infrequent and predictable (pp. 3-4) [36]. The second weakness of EPS-AKA was that the HN was not included as "...part of the authentication decision" (pp. 4) [36]. The authentication decision is made by the SN alone in 4G EPS-AKA (pp. 4) [36].

5G was introduced as a new generation aiming to provide improved network throughput and support for new technological use cases (e.g. IoT). Technical specifications for 5G included a proposed minimum peak data rate of 10 Gbps (uplink) and 20 Gbps (downlink) (pp. 2) [37]. The International Telecommunication Union Radiocommunication Sector (ITU-R) has defined three general use case scenarios for 5G: Enhanced Mobile Broadband (eMBB), Massive Machine-type Communications (mMTC), and Ultra-reliable and Low Latency Communications (URLLC) [38]. The eMBB scenario describes hotspots (higher data rates, user density, and traffic capacity) and seamless coverage (high mobility) [38]. The mMTC scenario is meant for Internet of Things (IoT), where many devices each require lower power consumption and data rate, respectively [38]. URLLC is meant for "...safety-critical and mission critical applications" [38].

In addition to improvements in network performance, 5G has also improved upon 4G authentication. 5G networks employ strong network authentication methods. One such method is 5G-AKA. In the 5G-AKA protocol, the UE always encrypts its permanent identity before it is sent to other entities in a 5G network (pp. 8) [36]. This addresses the vulnerability of UE identities under the EPS-AKA protocol. The 5G-AKA protocol also ensures that the HN makes the final decision for the authentication of a UE, instead of the SN (pp. 8) [36].

2.2 Development of 5G-AKA Requirements

The overall goal of our 5G-AKA (5G Authentication and Key Agreement) implementation is to create a protocol implementation developed in accordance with 3GPP specifications. Our project implementation should follow the scenario of a UE entity trying to authenticate itself with a 5G network endpoint. This scenario would be implemented purely in terms of cryptography. The authentication process (and associated cryptographic functions) would be implemented in accordance with 3GPP specifications. The implementation would not incorporate any other aspects of a 5G cellular network connection process (speed, distance from tower, etc.).

Our 5G-AKA implementation is meant to address the following question: would the proposed 3GPP 5G-AKA protocol successfully authenticate a user? The implementation also addresses another relevant question: could the 5G-AKA protocol be improved in any

way?

The implementation would need to clearly communicate the inner workings of each step of the 5G-AKA protocol. This would mean printing generated cryptographic values to a terminal console. The implementation would also utilize a single programming language. This would reduce development overhead. It would also demonstrate the simplicity of the authentication protocol.

2.3 Implementation of 5G-AKA Protocol

This section discusses the full technical specifications of the 5G-AKA protocol. It also discusses our Java implementation of the protocol.

2.3.1 Java Project Design and Structure

Our implementation plan was to develop three separate, concurrently-running processes that would represent the User Equipment, Serving Network, and Home Network entities. These processes would communicate with each other using TCP/IP sockets. These communications would represent the messages sent between network entities during the 5G-AKA authentication process.

Before starting development, decisions had to be made in terms of project design and structure. A primary concern was the development language to be used. We decided that the object-oriented Java programming language would be used as the programming language. The main justification for this choice was the abundance of cryptographic functions and importable libraries associated with the Java language. The java.base module contains the java.security and javax.crypto packages that were predicted to be of great utility to the project [39]. The java.security package offers support for cryptographic public key pair generation [40]. The javax.crypto package offers support for key generation, encryption, decryption, key agreement, and Message Authentication Code (MAC) generation [41].

The basic 5G-AKA authentication procedure consists of a short initiation phase, followed by 12 steps pertaining to the actual 5G-AKA protocol. These 12 steps are shown

(in overview) in Figure 2.1. The technical specifications of each step are discussed later in this Section 2.3.

Our Java implementation includes three Java files that correspond to the three entities shown in Figure 2.1. Our implementation has a single Java program that acts as the UE (`userEquipment.java` - 48 KB in size), another that acts as the SN (`servingNetwork.java` - 16 KB in size), and a third program that acts as the HN (`homeNetwork.java` - 33 KB in size).

Each of these three Java programs includes supporting Java files that contain functions used during authentication. For the HN this includes `hn_5gaka_he_av.java` and `hn_5gaka_init_auth.java`. For the SN this includes `misc_sn_functions.java`. For the UE this includes `ue_5gaka_he_av.java` and `ue_5gaka_init_auth.java`.

We developed our implementation using Java. Each of the three Java programs are compiled and run on a separate command-line terminal. The order of running the three files is as follows: run `homeNetwork.java` first, run `servingNetwork.java` second, and lastly run `userEquipment.java`.

We adopted a client-server architecture for the three running processes in our implementation. The HN is meant to behave (primarily) as a server. The SN is meant to behave as both a client to the HN, and as a server to the UE. The UE is meant to behave as a client. The HN and SN processes run before the UE process is started. The 5G-AKA protocol is initiated by the UE. Therefore, in our implementation, the HN and SN processes must be continuously running and waiting for the UE process to begin sending messages. In our implementation, the HN immediately sends a public key directly to the UE before the pre-5G-AKA initiation phase. The UE requires this public key as part of the pre-5G-AKA initiation phase.

In order to ensure that our code makes sense, we have relied on the protocol specifications outlined in 3GPP technical documentation. We have used this documentation as the basis for our 5G-AKA implementation, and to ensure that each step made sense as part of the overall authentication process. We use manual testing at different steps of our Java implementation to verify that we have implemented this authentication protocol correctly. This manual testing focuses on ensuring that outputted cryptographic values are correct.

Our Java implementation is an abstraction of the 5G-AKA protocol specified by the

3GPP. This is due to the fact that the SN is typically depicted as the *SEAF* in sequence diagrams, and the HN is broken down into two sub-entities: *AUSF* and *UDM/ARPF/SIDF*. Our 3-entity simplification is used because it streamlines the protocol implementation. In our implementation, the steps between the AUSF and UDM/ARPF/SIDF sub-entities are considered to be internal to the Home Network. Therefore, creating two separate Java programs for each sub-entity would be unnecessary for a successful proof-of-concept implementation of the 5G-AKA protocol. We discuss the SN and HN sub-entities below.

2.3.1.1 Serving Network Entity - SEAF

The SEcurity Anchor Function (SEAF) is a 5G Core network security entity (pp. 25) [1]. The SEAF is tasked with providing "...authentication functionality via the AMF in the serving network" (pp. 31) [1]. The SEAF will also "...support primary authentication using SUCI" (pp. 31) [1].

2.3.1.2 Home Network Entity - AUSF

The AUthentication Server Function (AUSF) is a 5G Core network security entity (pp. 25) [1]. It is a sub-entity of the HN that handles authentication, and informs the UDM of the status of subscriber authentications (pp. 32) [1].

2.3.1.3 Home Network Entity - UDM

The Unified Data Management (UDM) (pp. 23) [1] is used in conjunction with the ARPF to protect long-term keys (e.g. Home Network Private Key) from physical attacks (pp. 31) [1]. The UDM also leverages the SIDF to decrypt received SUCI values to determine the subscriber SUPI (pp. 32) [1]. The UDM also holds "...Home Network Public Key Identifier(s) for the private/public key pair(s) used for subscriber privacy" (pp. 32) [1].

2.3.1.4 Home Network Entity - ARPF

The Authentication credential Repository and Processing Function (ARPF) is a 5G Core network security entity (pp. 25) [1]. It is used in conjunction with the UDM to protect

long-term keys (e.g. Home Network Private Key) from physical attacks (pp. 31) [1]. The UDM/ARPF also generate authentication vectors in Step 1 of the 5G-AKA protocol (pp. 43) [1].

2.3.1.5 Home Network Entity - SIDF

The Subscription Identifier De-concealing Function (SIDF) is a 5G Core network security entity (pp. 25) [1]. It is a function that de-conceals the SUPI value from the SUCI value at the UDM sub-entity (pp. 98) [1]. This de-concealment is done using the HN private key value "...that is securely stored in the home operator's network to decrypt the SUCI" (pp. 98) [1].

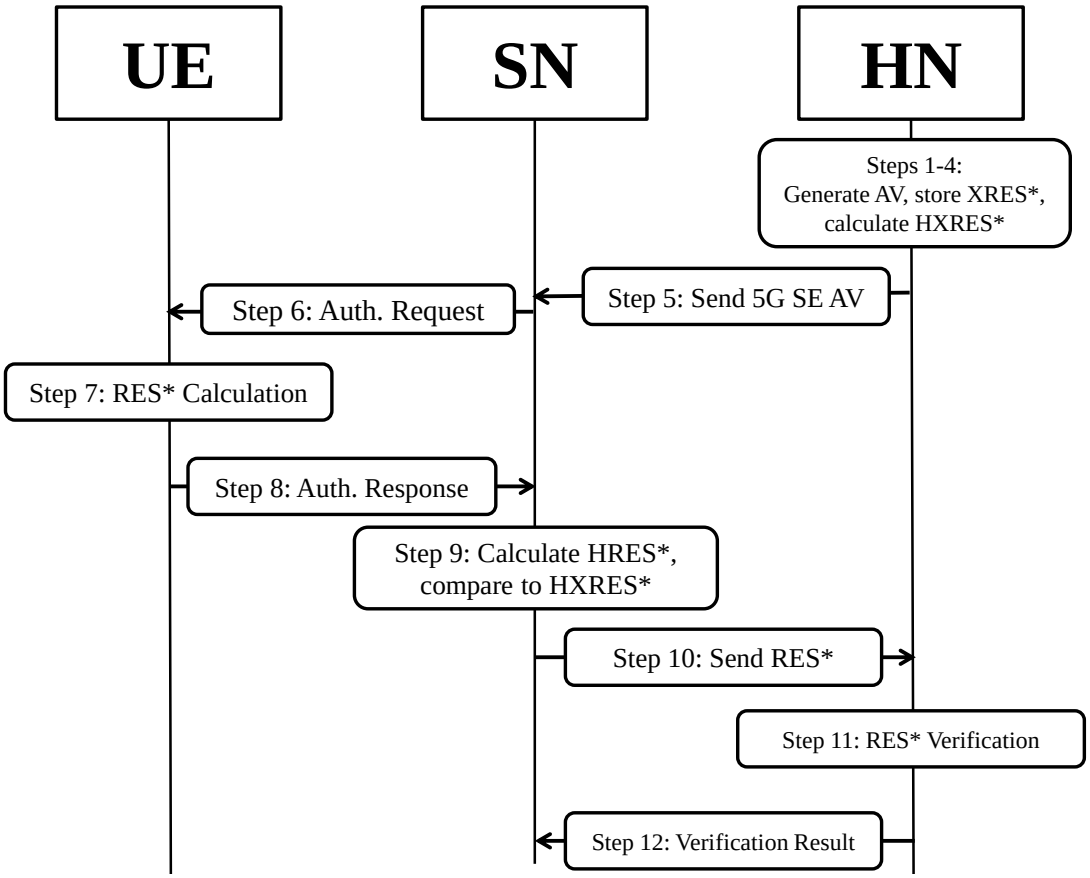

Figure 2.1: 5G-AKA Protocol Java Program UML Sequence Diagram

2.3.2 Pre-5G-AKA Initiation

Before the 5G-AKA authentication process is started, a short initiation phase must be completed first. This four-step phase is described as an initiation of the authentication procedure, and the selection of an authentication method (pp. 39) [1]. An overview of the initiation steps is shown in Figure 2.2.

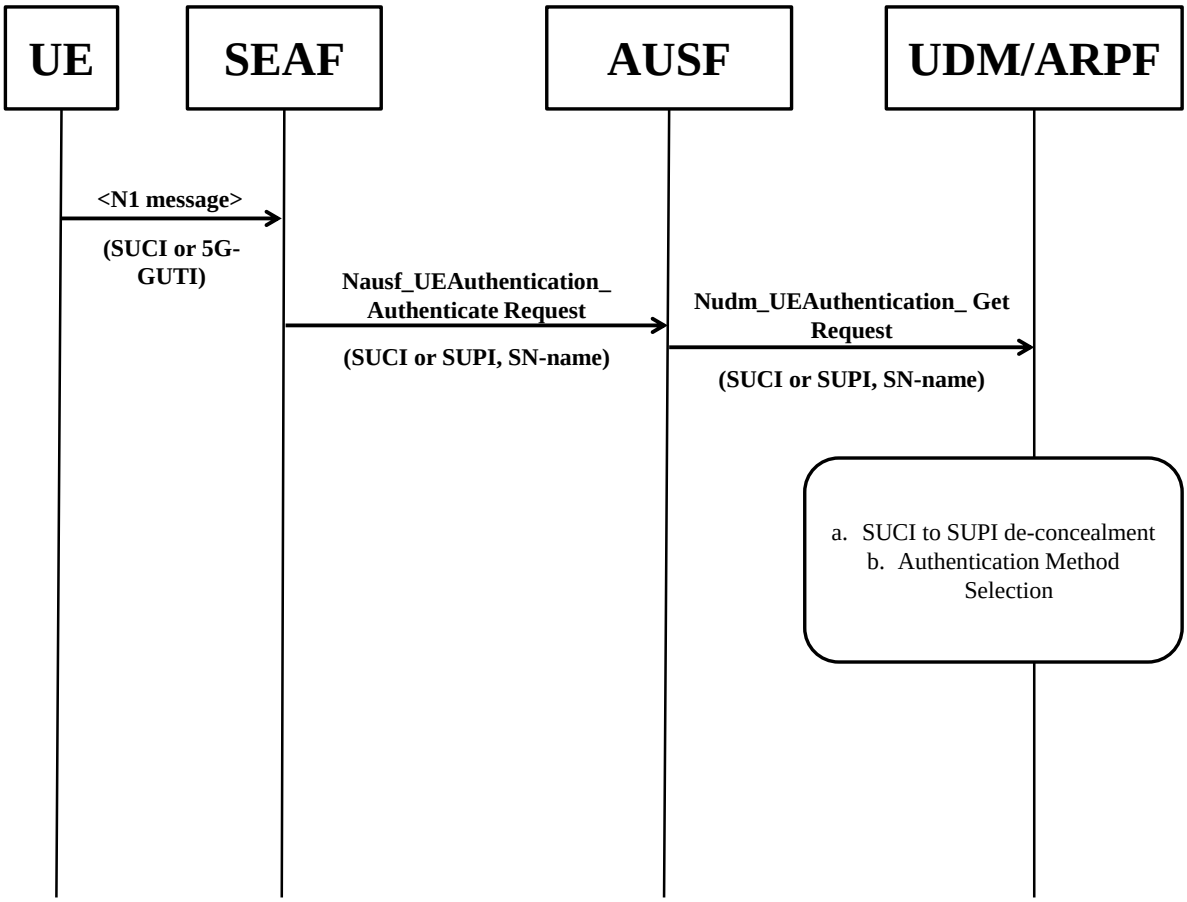

Figure 2.2: Primary Authentication and Authentication Method Selection UML Sequence Diagram, from (pp. 39) [1]

2.3.2.1 Pre-5G-AKA Initiation - Step 1

This initiation starts with the UE entity. The UE sends either a SUCI or 5G Global Unique Temporary Identifier (5G-GUTI) (pp. 22) [1] within an N1 message to the SEAF (SN) entity (pp. 39-40) [1].

Our implementation sends a SUCI value from the UE process to the SN process. For the N1 message formatting, a NAS-like message structure [42] is used. This is because Non-Access-Stratum (NAS) signalling and 5G-AKA primary authentication messages are both described as being a part of the "N1 Implementation" [42].

For the N1 NAS initiation step, the message structure shown in Figure 2.3 is used. For the extended protocol discriminator (EPD) 8-bit octet (octet 1), the value is set to *01111110*, signifying 5GS mobility management messages (pp. 105) [43]. For the security header type octet (octet 2), the value is set to *00000000*, denoting a non-security-protected plain 5GS NAS message (pp. 523) [2]. For the procedure transaction identity octet (octet 2a), the value is set to *00000000*, signifying that no specific PTI (Procedure Transaction Identity) has been assigned (pp. 108) [43]. For the message type (octet 3), the value is set to *01010110*. This value represents an authentication request (pp. 524) [2]. No further octets are used for additional information within the N1 message. In order to keep the N1 message formatting simple, the V (value-only) formatting is used for information elements (pp. 99) [43] such as the security header. The SUCI string value is appended to the N1 message before sending it to the SN Java process.

In our implementation, the N1-based message uses the following concatenation structure:

$$\text{N1 Message} := \text{“N1”} \| 01111110 \| 00000000 \| 00000000 \| 01010110 \| \text{SUCI}$$

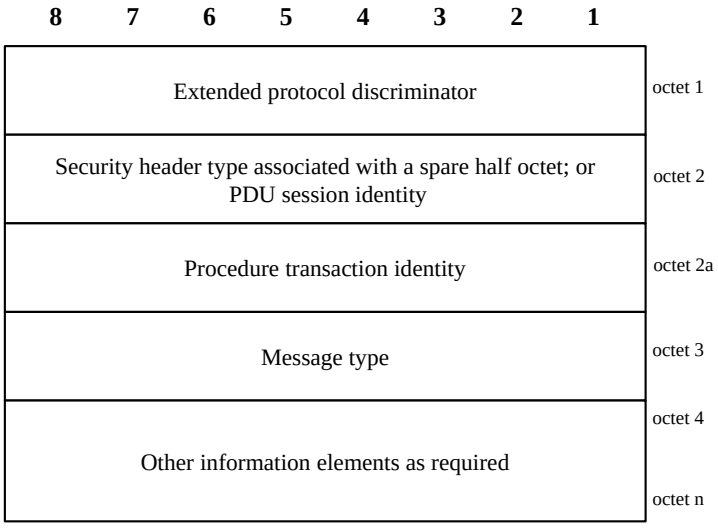

Figure 2.3: 5GS NAS message structure, from (pp. 521) [2]

2.3.2.2 SUCI

The Subscription Concealed Identifier (SUCI) (pp. 20) [3] is described as "...a privacy preserving identifier containing the concealed SUPI" (pp. 20) [3]. The structure of the SUCI is shown in Figure 2.4. In order to send a SUCI from the UE to the SN, there must be a pre-existing SUPI value from which the SUCI is derived.

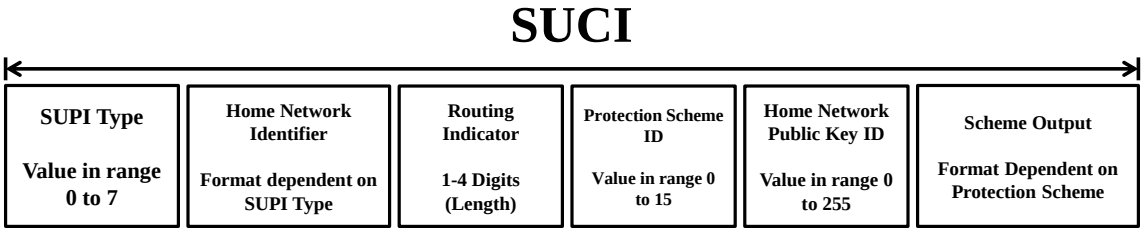

Figure 2.4: SUCI value structure, from (pp. 20) [3]

As seen in Figure 2.4, there are 6 essential components of the SUCI: the SUPI Type, Home Network Identifier, Routing Indicator, Protection Scheme ID, Home Network Public

Key ID, and the Scheme Output. We discuss each component in detail in the following pages.

2.3.2.2.1 SUPI Type The purpose of a Subscription Permanent Identifier (SUPI) (pp. 20) [3] is to be "...a globally unique 5G Subscription Permanent Identifier allocated to each subscriber in the 5G System" (pp. 20) [3]. The SUPI value consists of two parts: the SUPI type (value between 0 to 7, inclusive) and a SUPI value that corresponds to the type (pp. 20) [3]. Table 2.1 gives a full list of SUPI types.

SUPI Type Number	SUPI Type
0	IMSI
1	NAI
2	GLI
3	GCI
4-7	Not Yet Reserved

Table 2.1: SUPI Types, from (pp. 20-21) [3]

In our implementation, the SUPI type is "0", which corresponds to the IMSI. The structure of the International Mobile Subscription Identity (IMSI) (pp. 19) [3] value is shown in Figure 2.5. The three-digit Mobile Country Code (MCC) describes the country for a particular mobile subscription (pp. 21) [3]. The two-digit or three-digit Mobile Network Code (MNC) describes the home Public Land Mobile Network (PLMN) (pp. 50) [3] or Stand-alone Non-Public Network (SNPN) (pp. 52) [3] for a particular mobile subscription (pp. 21) [3]. The Mobile Subscriber Identification Number (MSIN) value identifies the mobile subscription pertaining to the PLMN or SNPN indicated by the MCC (pp. 20) [3].

We decided to use the IMSI SUPI type in our implementation. Although other SUPI types are available (e.g. GLI, GCI), 3GPP documentation provides test data set information for the "IMSI-based SUPI" (pp. 210-212) [1]. This provides guidelines for SUPI structuring and formatting.

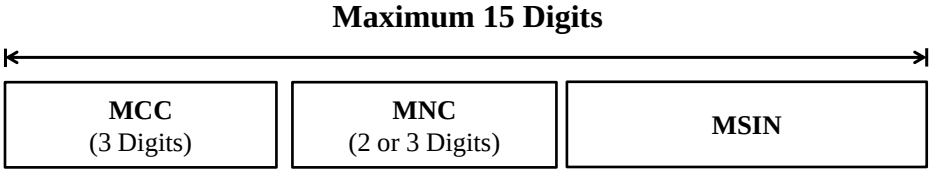

Figure 2.5: IMSI value structure, from (pp. 19) [3]

2.3.2.2.2 Home Network Identifier When the SUPI Type is set to "0" (IMSI), then the Home Network Identifier is simply a concatenation of the MCC and MNC (pp. 21) [3].

2.3.2.2.3 Routing Indicator The Routing Indicator is a value between 1 and 4 decimal digits in length (pp. 21) [3]. The Routing Indicator and the Home Network Identifier are used for routing the SUCI to available HN entities (pp. 21) [3].

2.3.2.2.4 Protection Scheme ID The Protection Scheme ID is a value between 0 and 15, inclusive. It describes the null or non-null protection scheme that applies to the SUCI value (pp. 21) [3]. A SUPI value is inputted, and a scheme output is generated based upon the Protection Scheme ID. Table 2.2 summarizes the existing protection scheme types.

Protection Scheme	Protection Scheme Identifier	Scheme Output Size
Null Scheme	0x0	Size of MSIN (if IMSI used)
Profile A	0x1	256-bit public key, 64-bit MAC, input size
Profile B	0x2	254-bit public key, 64-bit MAC, input size
Reserved	0x3 - 0xB	N/A

Table 2.2: Protection Scheme Types, from (pp. 211) [4]

The null scheme does not perform any encryption on the inputted SUPI. In the null scheme, the MSIN component of the IMSI is the scheme output (pp. 211) [4]. In our Java implementation, we include the entire IMSI as the input (and output) for the null scheme. This can be reduced to just the MSIN value in future iterations of our 5G-AKA

implementation.

In both Profiles A and B, Elliptic Curve Integrated Encryption Scheme (ECIES) is utilized for computing the SUCI from a SUPI value at the UE-side (pp. 212) [4]. At the HN-side, ECIES is used to de-conceal the SUPI value from the received SUCI value (pp. 213) [4].

The key differences between Profile A and B are in the Elliptic Curve (EC) Domain Parameters, EC Diffie-Hellman Primitive, and Point Compression. This is shown in Table 2.3.

The Key Derivation Function (KDF) (pp. 67) [44] for both Profile A and B is the ANSI-X9.63-KDF. This particular KDF is described as a "simple hash function construct" (pp. 32) [44]. The ANSI-X9.63-KDF function is structured as follows:

$$K = \text{ANSI-X9.63-KDF}(Z, \textit{keydatalen}, \textit{SharedInfo})$$

There are three inputs to the ANSI-X9.63-KDF. First, a shared secret octet string, denoted as Z (pp. 32) [44]. Second, the keying data length (in octets) is input using the integer value *keydatalen* (pp. 32) [44]. Thirdly, the optional octet string *SharedInfo* contains data shared across separate entities that seek to share Z (pp. 32) [44].

The ANSI-X9.63-KDF function has six steps (pp. 32-33) [44]. They are as follows:

ANSI-X9.63-KDF Step 1. The function verifies that the following inequality holds:

$|Z| + |SharedInfo| + 4 < hashmaxlen$ (pp. 32) [44].

If this inequality does not hold, then the ANSI-X9.63-KDF function stops (pp. 32) [44].

ANSI-X9.63-KDF Step 2. The function verifies that the following inequality holds:

$keydatalen < hashmaxlen \times (2^{32} - 1)$ (pp. 32) [44].

	Profile A	Profile B
EC Domain Parameters	Curve25519	secp256r1
EC Diffie-Hellman primitive	X25519	Elliptic Curve Cofactor Diffie-Hellman (ECCDH)
point compression	N/A	true
KDF	ANSI-X9.63-KDF	
Hash	SHA-256	
SharedInfo-1	Ephemeral Public Key Octet String - $\bar{R}$	
MAC	HMAC–SHA-256	
mackeylen	32 octets (256 bits)	
maclen	8 octets (64 bits)	
SharedInfo-2	The empty string	
ENC	AES–128 in CTR mode	
enckeylen	16 octets (128 bits)	
icblen	16 octets (128 bits)	
backwards compatibility mode	false	

Table 2.3: Protection Scheme Types, from (pp. 214-215) [4]

If this inequality does not hold, then the ANSI-X9.63-KDF function stops (pp. 32) [44].

ANSI-X9.63-KDF Step 3. The function initiates the 4 octet, big-endian string value *Counter* at 00000001_{16} (pp. 32) [44].

ANSI-X9.63-KDF Step 4. The function then initiates a for-loop from $i = 1$ to $i = \lceil keydatalen/hashlen \rceil$ (pp. 32) [44]. A pseudocode representation is shown in Algorithm 1.

Algorithm 1: ANSI-X9.63-KDF Step 4

Input: keydatalen, hashlen, Z, counter, SharedInfo
Output: Collection of K_{i} values
for $i = 1$ *to* $\lceil keydatalen/hashlen \rceil$ **do**
    $K_{\mathrm{i}} \leftarrow Hash(Z \| counter \| [SharedInfo])$;
    $counter \leftarrow counter + 1$;
    $i \leftarrow i + 1$;
end

ANSI-X9.63-KDF Step 5. The function sets K as the "leftmost *keydatalen* octets" (pp. 33) [44] of the following: $K_1 \| K_2 \| K_3 \| ... \| K_{\lceil keydatalen/hashlen \rceil}$.

ANSI-X9.63-KDF Step 6. The K value is outputted (pp. 33) [44].

2.3.2.2.5 Home Network Public Key Identifier The Home Network (HN) Public Key Identifier is a value between 0 and 255, inclusive (pp. 21) [3]. It holds the public key that identifies the SUPI protection key (pp. 21) [3]. This value can only be 0 if a null protection scheme is used (pp. 21) [3].

2.3.2.2.6 Scheme Output The Scheme Output can be a string of characters or hexadecimal digits, with variable length (pp. 21) [3]. It represents the output of the protection scheme used for the SUCI, outlined by the Protection Scheme ID (pp. 21) [3].

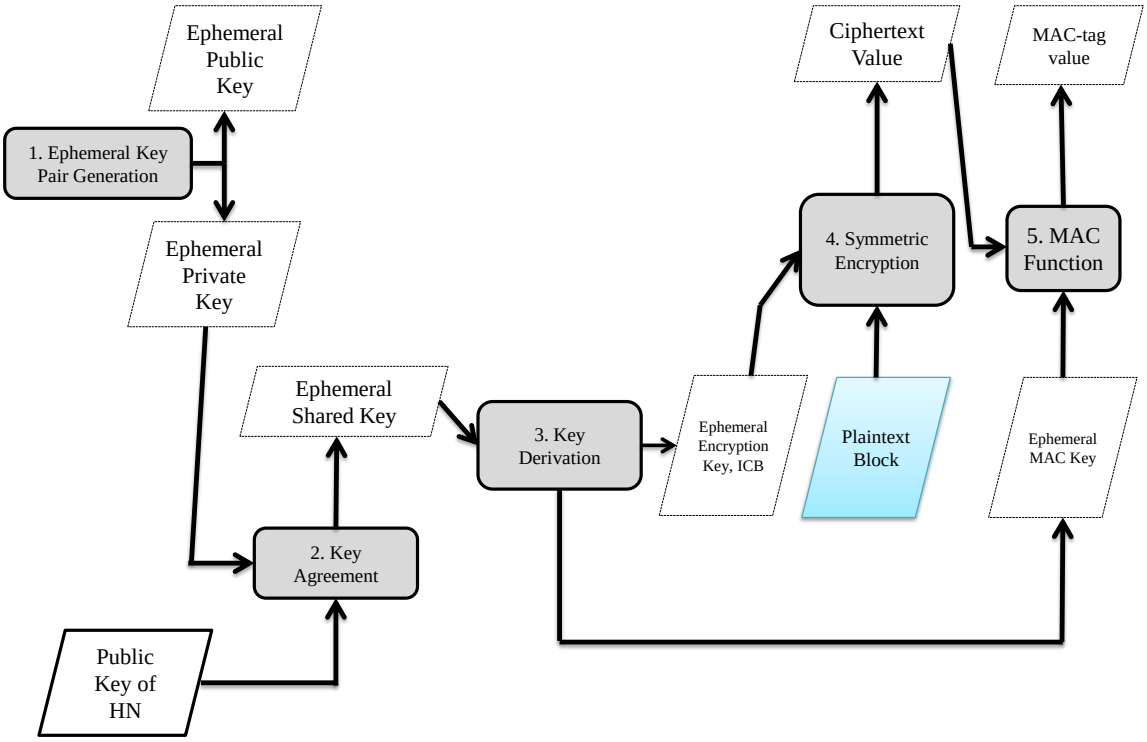

Figure 2.6: ECIES-based Encryption at UE-side, from (pp. 212) [4]

The encryption process of the SUPI value is shown in Figure 2.6. Each of the steps are outlined in further detail in the following pages. We include re-formatted code snippet versions of each function implementation for each step. These snippets illustrate the cryptographic calculations made at each step of the encryption process.

ECIES-based Encryption Step 1 - Ephemeral Key Pair Generation. This function generates the ephemeral public and private keys. The ephemeral public key will be used by the HN to de-conceal the SUPI from the received SUCI. It is part of the Scheme Output section of the SUCI. The private key is used as an input for the Key Agreement function in Step 2.

```
public KeyPair ephemeralKeyPairGeneration(String profileSelectionStr) {

      if ( profileSelectionStr  == "A") {

            KeyPairGenerator keyPairGeneratorEphem = null;

         try {

            keyPairGeneratorEphem = KeyPairGenerator.getInstance("X25519");

         } catch(NoSuchAlgorithmException x) {
            x.printStackTrace();
         }

         keyPairGeneratorEphem.initialize(255);
         KeyPair keyPairEphem = keyPairGeneratorEphem.generateKeyPair();
         this.ephKeyPairVal = keyPairEphem;
         return keyPairEphem;


      }

      else  if( profileSelectionStr  == "B"){

            KeyPairGenerator keyPairGeneratorEphem = null;

         try {
            keyPairGeneratorEphem = KeyPairGenerator.getInstance("EC");
            ECGenParameterSpec ECDomainParamSecp256r1 = new ECGenParameterSpec("secp256r1");
            keyPairGeneratorEphem.initialize(ECDomainParamSecp256r1);

         } catch(InvalidAlgorithmParameterException | NoSuchAlgorithmException x) {
            x.printStackTrace();
         }

         KeyPair keyPairEphem = keyPairGeneratorEphem.generateKeyPair();
         this.ephKeyPairVal = keyPairEphem;
         return keyPairEphem;
      }

      System.err.println("INVALID PROFILE SELECTION");
      this.ephKeyPairVal = null;
      return null;

   }
```

Figure 2.7: Ephemeral Key Pair Generation - Java Function [5]

In our implementation, the Java.security.KeyPairGenerator class is used to implement this particular function. This is shown in the code snippet of Figure 2.7.

In our implementation, the EC Domain Parameters are Curve25519 (Profile A) or secp256r1 (Profile B). The Curve25519 function is described as "...a state-of-the-art elliptic-curve Diffie-Hellman function suitable for a wide variety of cryptographic applications" (pp. 1) [45]. The term secp256r1 describes "...verifiably random elliptic curve domain parameters over $\mathbb{F}_p$..." (pp. 9) [46]. If an the inputted profile selection is invalid, an error message is printed to terminal, and the function returns a null value. The UE process will throw an exception when trying to extract the public and private keys from this null value. The code snippet in Figure 2.7 shows the implementation of the ephemeral key pair generation function. Code comments have been removed in this snippet for brevity. We have used the following citation in the development of this function: [5].

ECIES-based Encryption Step 2 - Key Agreement. The inputs to this function include the public key provided by the HN entity, and the newly generated ephemeral private key from Step 1. This function generates an ephemeral shared key.

In our implementation, we use the X25519 (Profile A) or ECCDH (Profile B) Diffie-Hellman primitive, based on the selected protection scheme. In order to implement a key agreement with ECCDH for Profile B, we use the BouncyCastle Java library. This is done using the following Java import:

```
import org.bouncycastle.jce.provider.BouncyCastleProvider;
```

BouncyCastle is required because ECCDH is not offered as part of standard Java cryptographic libraries. The code snippet in Figure 2.8 is the implementation of the key agreement function. Code comments have been removed in this snippet for brevity. We have used the following citations in the development of this function: [6] [7] [8] [9] [10] [11] [12].

```
public static KeyAgreement keyAgreementFunction(String profileSelectionStr, PrivateKey privKeyVal, PublicKey pubKeyVal) {

    if( profileSelectionStr .equals("A")) {

        try {

            KeyAgreement profileAKeyAgreement = KeyAgreement.getInstance("X25519");
            profileAKeyAgreement.init(privKeyVal);
            profileAKeyAgreement.doPhase(pubKeyVal, true);

            return profileAKeyAgreement;

        } catch(NoSuchAlgorithmException | InvalidKeyException x) {
            x.printStackTrace();
        }
    }

    else  if( profileSelectionStr .equals("B")){

        System.out.println(" Profile  B keyAgreementFunction");

        try {


            Provider BC = new BouncyCastleProvider();
            KeyAgreement profileBKeyAgreement = KeyAgreement.getInstance("ECCDH", BC);

            profileBKeyAgreement.init(privKeyVal);
            profileBKeyAgreement.doPhase(pubKeyVal, true);

            return profileBKeyAgreement;

        } catch(NoSuchAlgorithmException | InvalidKeyException x) {
            x.printStackTrace();
        }
    }

    return null;
}
```

Figure 2.8: Key Agreement - Java Function [6] [7] [8] [9] [10] [11] [12]

ECIES-based Encryption Step 3 - Key Derivation. The sole input value for this function is the ephemeral shared key. This function outputs an ephemeral encryption key, an Initial Counter Block (ICB) (pp. 213) [4], and an ephemeral MAC key. The code snippet in Figure 2.9 is the implementation of the key derivation function. The function follows the steps outlined for ANSI-X9.63-KDF earlier in this section. Some of the comments were removed for brevity. The following citations were used in the development of this function: [13] [14] [15] [16] [17] [18].

```
public static byte[] keyDerivationFn(byte[] kdfEphSharedSecretKeyVal, byte[] sharedInfoVal) {

      int keyDataLen = 128 + 128 + 256;
      double hashMaxLen = ((Math.pow(2, 64)) - 1) / 8;
      int hashLen = 256;

      int kdfEphSharedSecretKeyValLen = 8 * kdfEphSharedSecretKeyVal.length;
      int sharedInfoValLen = 0;

      boolean step1 = kdfEphSharedSecretKeyValLen + sharedInfoValLen + 4 < (hashMaxLen * 8) ;
      boolean step2 = keyDataLen < hashLen * ( (Math.pow(2, 32)) - 1 ) ;

      if ( step1 && step2 ) {

         double ceilRatio = Math.ceil(keyDataLen/hashLen);

         ByteArrayOutputStream outputK = new ByteArrayOutputStream();

         for(int i = 1; i <= ceilRatio; i++){

            String counterHexStrVal = Integer.toHexString(i);

            String concatStrZCounterSharedInfo1 = "";
            String ZStr = new String(kdfEphSharedSecretKeyVal, StandardCharsets.UTF_8);

            String SharedInfoStr = new String(sharedInfoVal, StandardCharsets.UTF_8);

            concatStrZCounterSharedInfo1 = ZStr + counterHexStrVal + SharedInfoStr;

            try {

               MessageDigest sha256_md = MessageDigest.getInstance("SHA-256");
               byte[] K_i = sha256_md.digest(concatStrZCounterSharedInfo1.getBytes(StandardCharsets.UTF_8));

               outputK.write(K_i);

            } catch (NoSuchAlgorithmException | IOException e) {
               e.printStackTrace();
            }


         }

         byte[] output_K_FINAL = outputK.toByteArray();

         return output_K_FINAL;

      }

      else {
         System.err.println("INVALID");
         return null;
      }
   }
```

Figure 2.9: Key Derivation Function - Java Function [13] [14] [15] [16] [17] [18]

```
public static String symmetricEncryptionFn(String ephEncrKey, byte[] icb, String plaintxtSUPI) {

    String salt = "abcd";

    try {

        IvParameterSpec ivspec = new IvParameterSpec(icb);

        SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
        KeySpec spec = new PBEKeySpec(ephEncrKey.toCharArray(), salt.getBytes(), 16384, 128);
        SecretKey tmp = factory.generateSecret(spec);
        SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");

        Cipher cipher = Cipher.getInstance("AES/CTR/NoPadding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivspec);
        return Base64.getEncoder().encodeToString(cipher.doFinal(plaintxtSUPI.getBytes("UTF-8")));
    }

    catch (Exception e)
    {
        System.out.println("SYMMETRIC ENCRYPTION ERROR: " + e.toString());
    }
    return null;

}
```

Figure 2.10: Symmetric Encryption Function - Java Function [19] [20] [21]

ECIES-based Encryption Step 4 - Symmetric Encryption. The Plaintext Block (containing the SUPI) undergoes symmetric encryption using the ephemeral encryption key and ICB generated in step 3 of ECIES-based encryption. The outputted Ciphertext value will become part of the Scheme Output of the SUCI.

The symmetric encryption specification is AES-128 in CTR mode. As shown in Figure 2.10, the key length is set to 128 bits. In our implementation, the SecretKeySpec and Cipher are set in accordance with the specification as well. A very simple string value is used for the salt in our implementation. Since the implementation is a proof-of-concept for the 5G-AKA protocol, this basic salt value serves its purpose. For deployment in a real 5G network environment, a more sophisticated salt value would be needed.

The code snippet in Figure 2.10 is our implementation of the symmetric encryption function. Some of the comments have been removed for brevity. We have used the following citations in the development of this function: [19] [20] [21].

```
public static String MACFunction(String cipherTextVal, String ephMACKey) {

      Mac sha256Hmac;
      String mac_tag;

      try{

          final byte[] byteKey = ephMACKey.getBytes(StandardCharsets.UTF_8);

          sha256Hmac = Mac.getInstance("HmacSHA256");

          SecretKeySpec keySpec = new SecretKeySpec(byteKey, "HmacSHA256");

          sha256Hmac.init(keySpec);

          byte[] macData = sha256Hmac.doFinal(cipherTextVal.getBytes(StandardCharsets.UTF_8));
          mac_tag = Base64.getEncoder().encodeToString(macData);

          return mac_tag;

      } catch (NoSuchAlgorithmException | InvalidKeyException e) {
          e.printStackTrace();
      }


      return null;

  }
```

Figure 2.11: MAC - Java Function [22]

ECIES-based Encryption Step 5 - MAC Function. The newly generated Ciphertext and the ephemeral MAC key are inputted into the MAC function to generate a MAC-tag value for authentication at the HN-side during de-concealment. The code snippet in Figure 2.11 is our implementation of the MAC function. Some of the comments have been removed for brevity. We have used the following citation in the development of this function: [22].

The final output of the UE-side encryption process in Figure 2.6 is a concatenation of the ephemeral public key, Ciphertext, and MAC-tag value. This output constitutes the Scheme Output value of the SUCI. The SUCI is sent from the UE to the SN as the first 5G-AKA initiation step. The Scheme Output concatenation is shown below:

$$\text{Scheme Output} = \text{ephemeral public key} \| \text{Ciphertext} \| \text{MAC}$$

2.3.2.3 Pre-5G-AKA Initiation - Step 2

Step 1 has been completed, and the SUCI has been sent from the UE to the SN. Upon receipt of the SUCI value, the SN then sends the received SUCI (paired with the SN-name) to the AUSF sub-entity of the HN within a Nausf_UEAuthentication_Authenticate Request (pp. 39-40) [1]. Alternatively, the SN could send a SUPI instead, requiring it to decode any SUCI received from a UE entity (pp. 39-40) [1]. Our implementation has the SN send the SUCI value.

During initiation, a message of the Nausf_UEAuthentication_Authenticate Request format is required to have a SUPI (or SUCI) and SN name as input (pp. 181) [1]. The output is required to be an authentication vector (pp. 181) [1].

2.3.2.4 Pre-5G-AKA Initiation - Step 3

Once the AUSF has received the message from the SEAF, it forwards the message (unchanged) to the UDM/ARPF/SIDF (also part of the HN) (pp. 39-40) [1]. The message is sent in Nudm_UEAuthentication_GetRequest format.

During initiation, a message of the Nudm_UEAuthentication_GetRequest format is required to have a SUPI (or SUCI) and SN name as input (pp. 183) [1]. The output is required to be an authentication method (pp. 183) [1]. Another output will be the SUPI, since SUCI is used as input in the Java implementation (pp. 183) [1].

2.3.2.5 Pre-5G-AKA Initiation - Step 4a

The UDM/ARPF/SIDF entity performs de-concealment to extract the SUPI from the SUCI (if a SUCI was sent originally) (pp. 39-40) [1]. This is a process of ECIES-based decryption.

In our implementation, we included a simple manual test that outputs the decrypted SUPI value to the command-line terminal running the Home Network process. This was done to verify that the extracted SUPI value matches the explicitly-defined SUPI value in the User Equipment process.

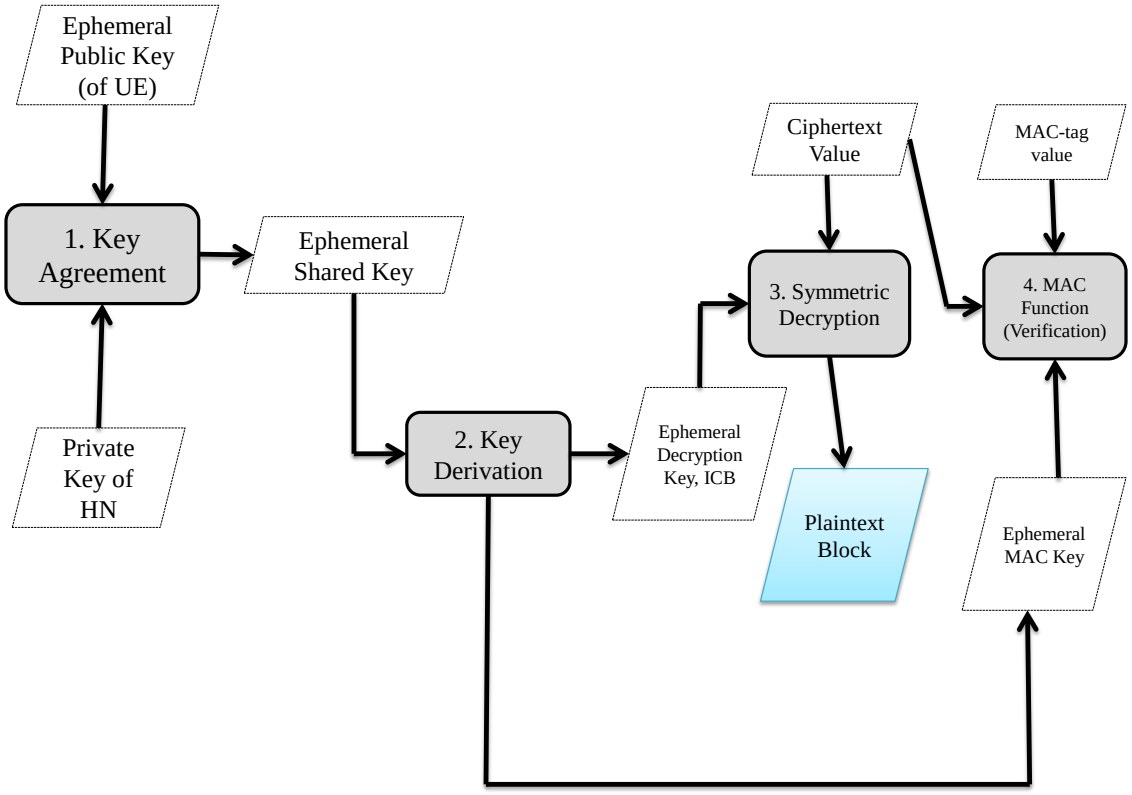

Figure 2.12: ECIES-based Decryption at UE-side, from (pp. 213) [4]

The decryption process of the SUCI value is shown in Figure 2.12. Each of the steps are outlined in further detail as follows:

ECIES-based Decryption Step 1 - Key Agreement. The inputs to this function include the ephemeral public key of the UE and the private key of the HN. The function outputs an ephemeral shared key. The general structure of the function used in this step is the same as the key agreement function for encryption.

ECIES-based Decryption Step 2 - Key Derivation. The ephemeral shared key is the sole input to this function. The output consists of an ephemeral MAC Key, and ephemeral decryption key, and an ICB. The general structure of the function used in this step is the same as the key derivation function for encryption.

ECIES-based Decryption Step 3 - Symmetric Decryption. The inputs include the received Ciphertext value (from the UE), the Ephemeral decryption key, and ICB. The function outputs a plaintext block. If the decryption process is successful, then this plaintext block should be the de-concealed SUPI. Due to symmetric encryption and decryption, the key used in this function is of the same value as the ephemeral encryption key derived at the UE during encryption.

ECIES-based Decryption Step 4 - MAC Function (Verification). The inputs include the received Ciphertext value (from the UE), the ephemeral MAC Key, and the received MAC-tag value (from the UE). This function generates a new MAC-tag value and compares it against the received MAC-tag value. If they are equal in value, then the decryption is considered verified and successful.

2.3.2.6 Pre-5G-AKA Initiation - Step 4b

At this step, the de-concealment process at the HN-side is complete. In our implementation, the de-concealment is successful. We ensure this through manual testing. Our HN process outputs the de-concealed SUPI, and it is verified to be the same as the original SUPI defined in the UE process.

In the final step of the initiation, the UDM/ARPF/SIDF selects a method of authentication (pp. 39-40) [1]. In our implementation, the selection is the 5G-AKA authentication protocol.

2.3.3 5G-AKA Authentication Protocol

Once the initiation steps are completed without error, the 5G-AKA protocol steps are initiated by the HN entity. The full 5G-AKA protocol is shown in Figure 2.13. The remainder of this section is a summary of the 5G-AKA authentication protocol steps. We discuss the specifications of each step (outlined by the 3GPP), and how our implementation follows these requirements.

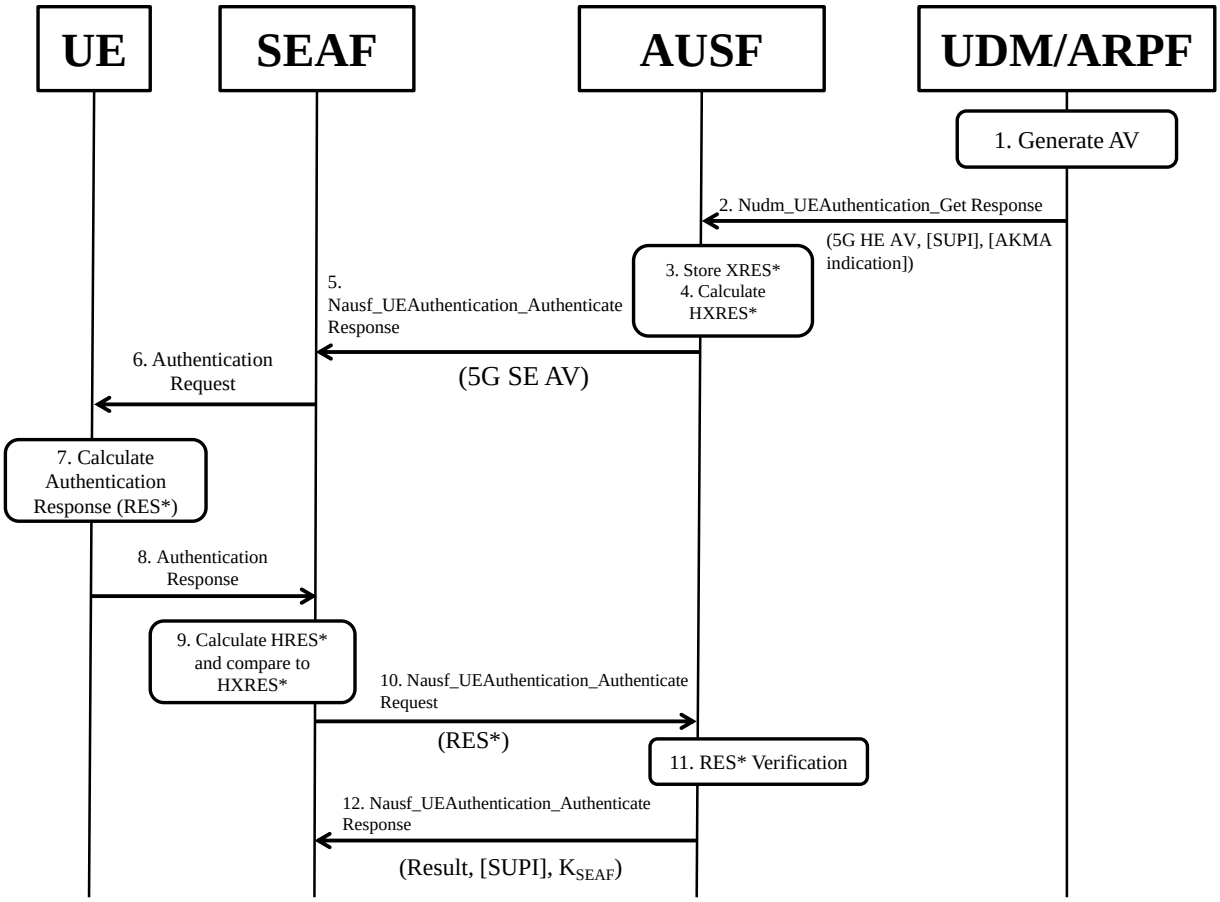

Figure 2.13: 5G-AKA Authentication Protocol UML Sequence Diagram, from (pp. 43) [1]

2.3.3.1 5G-AKA - Step 1

First, the UDM/ARPF creates a 5G HE AV (pp. 43) [1]. The properties of this AV include the Access and Mobility Management Function (AMF) separation bit being set to "1" (pp. 43) [1]. The AMF is a 16-bit string with bits labelled from "0" (Most Significant Bit) to "15" (Least Significant Bit) (pp. 73) [23]. Hence, the AMF value is "1000000000000000". A K_{AUSF} is derived, and an XRES* value is calculated (pp. 43) [1]. K_{AUSF} is an "intermediate key" (pp. 37) [1].

The 5G HE AV is created using the following values: A randomly-generated value

(RAND), XRES*, Authentication Token (AUTN), and K_{AUSF} (pp. 43) [1].

The RAND represents a random integer value. In our implementation, an instance of java.util.Random is used to generate a value between 0 and 147483637.

The XRES and XRES* values represent an "eXpected RESponse" (pp. 23) [1]. In order to calculate the XRES* value, first the Expected Response (XRES) value must be generated. A message authentication function (HMAC-SHA-256) is used to generate XRES, as shown in Figure 2.14. The inputs to this MAC function are the HN public key and the RAND value.

Functions *f1* and *f2* are MAC functions (pp. 11) [23]. Functions *f3*, *f4*, and *f5* are key generation functions that compute Cipher Key (CK), Integrity Key (IK), and Anonymity Key (AK), respectively (pp. 11) [23].

Parameter Name	Value
FC	0x6B
P0	Serving Network Name
L0	Length of Serving Network Name
P1	RAND
L1	Length of RAND
P2	XRES
L2	Length of XRES

Table 2.4: XRES* S Parameters, from (pp. 192) [1]

The XRES* value is derived from an input S value and input key KEY. The XRES, RAND, and the SN name values are assigned to parameter labels, shown in Table 2.4. The parameters in Table 2.4 are concatenated to form the S value as follows (pp. 40) [47]:

$$S = FC \| P0 \| L0 \| P1 \| L1 \| P2 \| L2$$

The KEY value is a concatenation of the CK and IK values as follows (pp. 192) [1]:

$$KEY = CK \| IK$$

The S value and input key KEY are inputted into a HMAC-SHA-256 KDF function to generate the XRES* value.

The next component of the 5G HE AV is the AUTN. The AUTN value is derived using the concatenation formula shown below (pp. 22) [23]. The $\oplus$ symbol represents the Exclusive-or operation (XOR).

$$\text{AUTN} := (\text{SQN} \oplus \text{AK}) \| \text{AMF} \| \text{MAC}$$

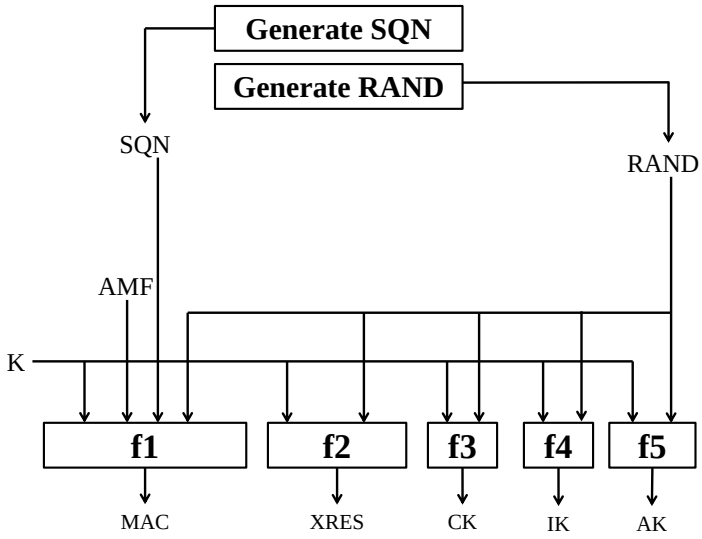

Figure 2.14: Generation of MAC, XRES, and Keys, from (pp. 22) [23]

The K_{AUSF} is derived using a KDF with two inputs: an input S and an input key labelled KEY. The KEY value is a concatenation of the CK and IK values, as follows (pp. 191) [1]:

$$KEY = CK \| IK$$

The S value is a concatenation of parameters shown in Table 2.5. The parameters in Table 2.5 are concatenated to form the S value as follows (pp. 40) [47]:

$$S = FC \| P0 \| L0 \| P1 \| L1$$

Parameter Name	Value
FC	0x6A
P0	Serving Network Name
L0	Length of Serving Network Name
P1	SQN $\oplus$ AK
L1	Length of SQN $\oplus$ AK

Table 2.5: K_{AUSF} S Parameters, from (pp. 191) [1]

Our Java implementation uses the HMAC-SHA-256 function for the KDF in order to derive the K_{AUSF}. We structure the 5G HE authentication vector using the following concatenation formula (pp. 22) [23]:

$$\text{AV} := \text{RAND}\|\text{XRES*}\|K_{\text{AUSF}}\|\text{AUTN}$$

At this stage step 1 is complete. This authentication vector is sent to another sub-entity of the HN in step 2.

2.3.3.2 5G-AKA - Step 2

The UDM/ARPF sub-entity will return the 5G HE AV (to the AUSF), specifying that 5G-AKA is the chosen authentication method (pp. 43) [1]. The SUPI value is also returned if a SUCI was sent in the initiation phase (pp. 43) [1]. Also, if the subscriber has an AKMA subscription, an indication value for AKMA will be included in the response (pp. 44) [1].

The message in this step is of Nudm_UEAuthentication_Get Response format. In our implementation, there is a flag to indicate the existence of an AKMA subscription. However, this flag is never used as it is not integral to the 5G-AKA authentication process. This value could be used in future iterations of the 5G-AKA protocol if the AKMA subscription plays a more prominent role in authentication.

2.3.3.3 5G-AKA - Step 3

The AUSF sub-entity of the HN temporarily stores the XRES* value along with the SUCI or SUPI that the UDM/ARPF has sent (pp. 44) [1]. The XRES* is stored in order to be used later in Step 11 of the 5G-AKA protocol.

2.3.3.4 5G-AKA - Step 4

The AUSF then generates a 5G AV fom the received 5G HE AV (pp. 44) [1]. This is done through the following calculations: The AUSF computes an HXRES* value from XRES*, and a new K_{SEAF} value from K_{AUSF} (pp. 44) [1]. These new values replace their respective progenitors in the original 5G HE AV (pp. 44) [1], shown in Figure 2.15.

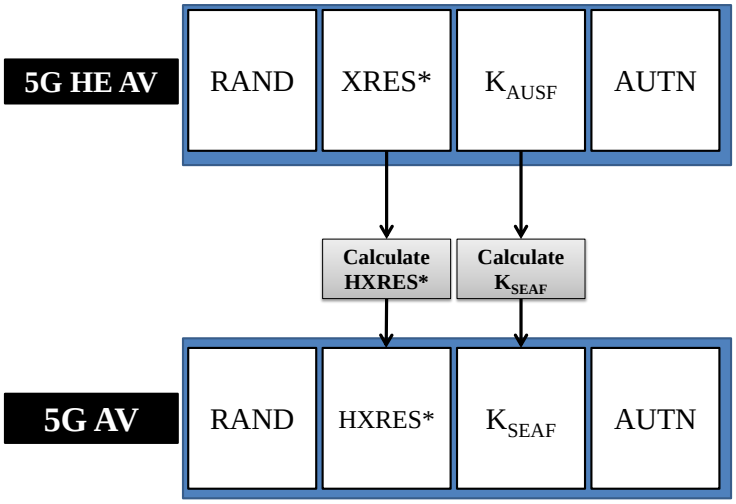

Figure 2.15: 5G AV Derivation

The term HXRES stands for "Hash eXpected RESponse" (pp. 22) [1]. HXRES* is derived from XRES* with a SHA-256 hashing algorithm using the input S value (pp. 192) [1] shown below. The values of the parameters *P0* and *P1* are shown in Table 2.6.

$$S = P0 \| P1$$

Parameter Name	Value
P0	RAND
P1	XRES*

Table 2.6: HXRES* S Parameters, from (pp. 192) [1]

$\mathrm{K_{SEAF}}$ is an intermediate key value derived from $\mathrm{K_{AUSF}}$ via a KDF using the input S and KEY values shown below (pp. 192) [1]. The values of the parameters in these derivations are shown in Table 2.7.

$$S = FC \| P0 \| L0$$

$$KEY = \mathrm{K_{AUSF}}$$

Parameter Name	Value
FC	0x6C
P0	Serving Network Name
L0	Length of Serving Network Name

Table 2.7: $\mathrm{K_{SEAF}}$ S Parameters, from (pp. 192) [1]

Our implementation uses the HMAC-SHA-256 function for the KDF in order to derive the $\mathrm{K_{SEAF}}$.

2.3.3.5 5G-AKA - Step 5

The AUSF sub-entity removes the K_{SEAF} value from the 5G HE AV, and sends a 5G SE AV to the SEAF sub-entity of the SN (pp. 44) [1]. The 5G SE AV contains the RAND, AUTN, and HXRES* values (pp. 44) [1]. This process is shown in Figure 2.16.

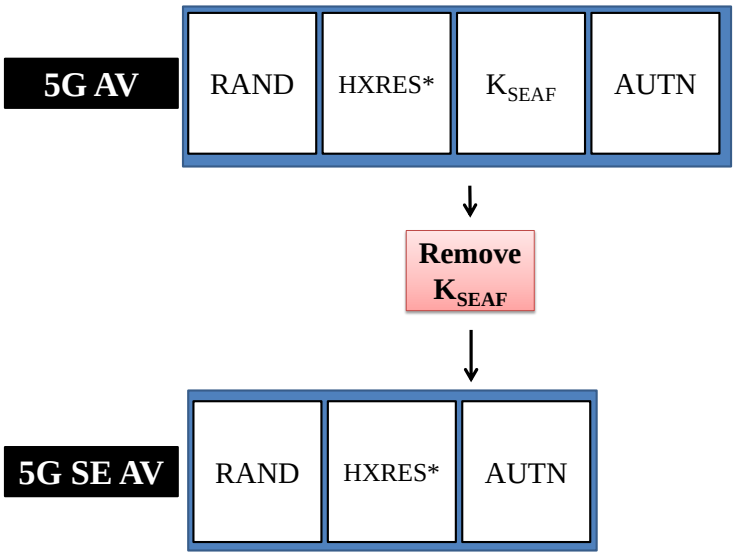

Figure 2.16: 5G SE AV Generation

2.3.3.6 5G-AKA - Step 6

The SEAF sub-entity sends the RAND and AUTN values to the UE entity within a NAS-format Authentication-Request message (pp. 44) [1]. This message also includes the 5G Key Set Identifier (ngKSI) (pp. 22) [1] and Anti-Bidding down Between Architectures (ABBA) (pp. 21) [1] parameters (pp. 44) [1]. The UE and AMF use the ngKSI value in order to "...identify the K_{AMF} and the partial native security context that is created if the authentication is successful" (pp. 44) [1]. This identification of the the K_{AMF} occurs *after* the completion of the 12 steps of the 5G-AKA protocol. It is not integral to our 5G-AKA authentication protocol implementation itself.

The ngKSI within the Authentication-Request message is a NAS key set identifier (pp. 464) [2]. Our implementation structures the ngKSI as a 3-bit binary value (pp. 577) [2], with a placeholder of "110". The ABBA parameter is used by the UE for K_{AMF} derivation (pp. 193) [1]. The default ABBA value is "0x0000" (pp. 193) [1].

2.3.3.7 5G-AKA - Step 7

The freshness of the 5G AV value is verified based on the AUTN value (pp. 44) [1]. If the AV is considered fresh, the Response value (RES) is calculated (pp. 44) [1]. Freshness verification is a two-step process (pp. 24) [23]. First, there is a comparison of MAC values (pp. 24) [23]. Second, there is a comparison of SQN values (pp. 24) [23]. The process of verifying freshness is discussed in further detail in Section 2.3.4. This freshness verification serves as another instance of manual testing used to ensure our implementation has correct cryptographic values.

The UE then calculates the RES* value from the RES value (pp. 44) [1]. The K_{AUSF} and K_{SEAF} intermediate keys are also calculated (pp. 44) [1]. The separation bit in the AMF field of the AUTN value is verified to be set to 1 (pp. 44) [1].

In order to calculate the RES* value, first the RES value must be generated. A message authentication function is used to generate RES, as shown in Figure 2.17.

Parameter Name	Value
FC	0x6B
P0	Serving Network Name
L0	Length of Serving Network Name
P1	RAND
L1	Length of RAND
P2	RES
L2	Length of RES

Table 2.8: RES* S Parameters, from (pp. 192) [1]

The RES* value is derived from an input S value and input key KEY. The RES, RAND, and the SN name values are used in the formation of S as shown in Table 2.8. The parameters in Table 2.8 are concatenated to form the S value as follows (pp. 40) [47]:

$$S = FC \| P0 \| L0 \| P1 \| L1 \| P2 \| L2$$

The input key KEY is derived as follows (pp. 192) [1]:

$$KEY = \text{CK} \| \text{IK}$$

The K_{AUSF} value is derived in the exact same manner as was done in Step 1 of the 5G-AKA protocol. The parameters are shown in Table 2.5. The AK is calculated at the UE-side using the HN public key and the RAND value received from the SN. This is shown in Figure 2.14. The SQN value used is originally generated by the UE-entity. Hence, in this context, it is referred to as SQN_{UE}. The K_{SEAF} value is derived in the exact same manner as was done in step 4 of the 5G-AKA protocol. The parameters are shown in Table 2.7.

2.3.3.8 5G-AKA - Step 8

The UE returns the RES* value to the SEAF sub-entity via a NAS-format authentication response message (pp. 44) [1].

2.3.3.9 5G-AKA - Step 9

The SEAF computes HRES* from RES*, and compares this HRES* value to the HXRES* calculated in step 4 (pp. 44) [1]. HRES* is derived from RES* with a SHA-256 hashing algorithm using the input S value (pp. 192) [1] shown below. The values of the parameters *P0* and *P1* are shown in Table 2.9.

$$S = P0 \| P1$$

Parameter Name	Value
P0	RAND
P1	RES*

Table 2.9: HRES* S Parameters, from (pp. 192) [1]

Authentication is considered successful if HRES* and HXRES* coincide (i.e. they are equal in value) (pp. 44) [1]. Error handling must be initiated if they do not coincide (p. 45) [1]. This check for equality between HRES* and HXRES* serves as a manual test for the correctness of our implementation.

In our implementation, the result of this coincidence-check is outputted to the command-line terminal running the Serving Network process. This is a manual test to verify the equality of HRES* and HXRES*.

2.3.3.10 5G-AKA - Step 10

The SEAF sends the RES* value to the AUSF sub-entity of the HN (pp. 44) [1]. The format of this message is that of a Nausf_UEAuthentication_Authenticate Request (pp. 44) [1].

2.3.3.11 5G-AKA - Step 11

Upon receipt of the message from the SEAF, the AUSF checks for AV expiry (pp. 44) [1]. We presume that the criteria for expiry is based upon the time of generation of the AV. If expired, the authentication may be considered to be unsuccessful (pp. 44) [1]. If not expired, the K_{AUSF} is stored at the AUSF (pp. 44) [1]. In our implementation, we do not include this expiry-check since it is not considered a critical component of the proof-of-concept.

After the expiry check, the AUSF entity then verifies whether the XRES* and received RES* values are equal. If so, then "...the AUSF shall consider the authentication as successful from the home network point of view" (pp. 44) [1]. The UDM sub-entity is updated

if this happens (pp. 44) [1].

In our implementation, the result of this coincidence-check is outputted to the command-line terminal running the Home Network process. This is a manual test to verify the equality of RES* and XRES*.

2.3.3.12 5G-AKA - Step 12

The AUSF sends a returning message to the SEAF, one that indicates successful or unsuccessful authentication from the point-of-view of the HN entity (pp. 44) [1]. Upon successful authentication, this returning message will contain the K_{SEAF} value (pp. 44) [1]. If a SUCI is used as part of the pre-5G-AKA initial authentication request, then the AUSF will include the SUPI value as part of its final returning message to the SEAF (pp. 44) [1]. The format of this message is that of a Nausf_UEAuthentication_Authenticate Response (pp. 44) [1].

2.3.4 5G-AKA Error Handling

The 5G-AKA protocol has outlined specific procedures for handling two types of authentication failures: Synchronization failures and MAC failures. These failures can be detected at the UE network entity, specifically the USIM sub-entity (pp. 45) [1]. The detection of these errors is performed at Step 7 (pp. 45) [1] of the 5G-AKA protocol (see figure 2.13).

Step 7 is concerned with verifying Authentication Vector (AV) freshness by testing the freshness of the received AUTN value. This requires two verification steps, outlined below (pp. 24) [23] (pp. 1387) [24]:

1. Verify that MAC = XMAC (MAC Failure Check).
2. Verify that the SQN value is within the acceptable range (Synchronization Failure Check).

The derivation for the values required for these verification steps are shown in Figure 2.17. These derivations are performed at the UE-side. The AUTN is originally derived in

step 1 of the 5G-AKA protocol. The boxes labelled *f1*, *f2*, *f3*, *f4*, and *f5* in Figure 2.17 each represent different cryptographic functions.

The function *f1* is a "message authentication function" that computes a MAC value (pp. 11) [23], specifically the XMAC value.

The function *f2* is another "message authentication function" that computes RES (and XRES) values (pp. 11) [23].

The functions *f3*, *f4*, and *f5* are "key generating functions" used to compute the CK, IK, and AK values, respectively (pp. 11) [23]. These three functions are templates of the KDF used in the concealment of SUPI values in 5G-AKA primary authentication shown in step 1 of Figure 2.2.

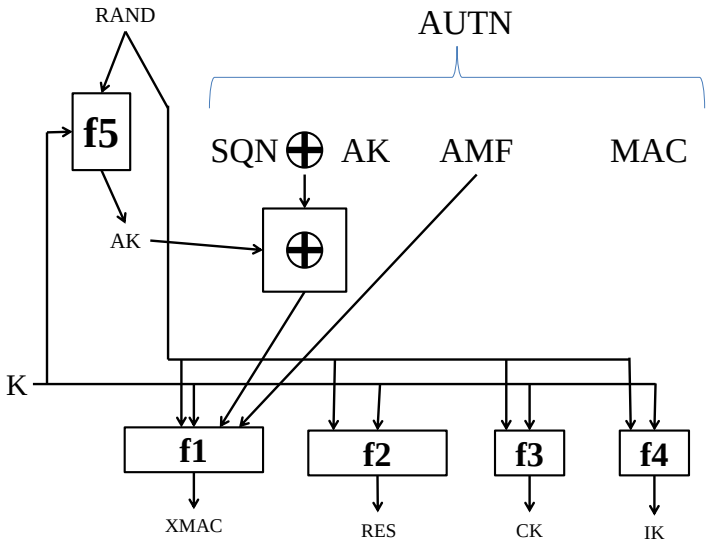

Figure 2.17: 5G-AKA AUTN Freshness Verification Operations, from (pp. 24) [23]

If the MAC Failure Check finds that MAC $\neq$ XMAC, then a simple "MAC Failure" message is sent from the UE to the SN (pp. 1387) [24].

The second AUTN verification step is concerned with the two separate SQN values that belong to the UE and HN entities, respectively. In order to perform this particular verification step, the parameters of the "acceptable range" have to be clearly defined first. The SQN that is defined at the HN (referred to as "SQN_{HN}") is given a value of 1 in our implementation. It is represented as a 48-bit binary string with the least significant bit set to "1", and all remaining bits set to "0". The SQN that is defined at the UE (referred to as "SQN_{UE}") is given a value of 0 in our implementation. It is represented as a 48-bit binary string comprised entirely of bits set to "0". We set these values for SQN_{HN} and SQN_{UE} based on the example in citation [48].

According to a paper titled *A Formal Analysis of 5G Authentication* by Basin et al., an acceptable range requirement would be the following (pp. 1387) [24]:

$$\text{SQN}_{\text{UE}} < \text{XSQN}_{\text{HN}}$$

In this statement, we define XSQN_{HN} as the following (pp. 1387) [24]:

$$\text{XSQN}_{\text{HN}} = \text{AK} \oplus (\text{SQN}_{\text{HN}} \oplus \text{AK})$$

If the Synchronization Failure Check finds that the SQN_{UE} is not within the defined acceptable range, then the SN and HN entities are notified of this. An AUTS value is generated by the UE. The derivation of the AUTS value is shown below (pp. 24-25) [23] (pp. 1387) [24]. In our implementation, the f1* function uses an HMAC-SHA-256 hashing function to generate the MAC-S value. The f5* function is the same ANSI-X9.63-KDF used to generate an AK value, as discussed in Step 1 of the 5G-AKA protocol.

$$\text{AUTS} := \text{Conc}(\text{SQN}_{\text{UE}}) \| \text{MAC-S}$$

$$\text{Conc}(\text{SQN}_{\text{UE}}) := \text{SQN}_{\text{UE}} \oplus \text{f5*}(\text{K, RAND})$$

$$\text{MAC-S} := \text{f1*}(\text{ K, SQN}_{\text{UE}} \| \text{RAND} \| \text{AMF})$$

The AUTS is sent from the UE to the SN. At the SN, the RAND (random value) and SUCI values are attached to the message. The message is then sent to the HN. If the MAC Failure Check passes (MAC-S is equal to xMAC) but the Synchronization Failure Check does not (at the HN), then the SQN_{HN} is updated to explicitly be of a value greater than SQN_{UE}. This process is shown in Figure 2.18.

In our implementation, once the SQN_{HN} is updated, the entire 5G-AKA protocol starts again.

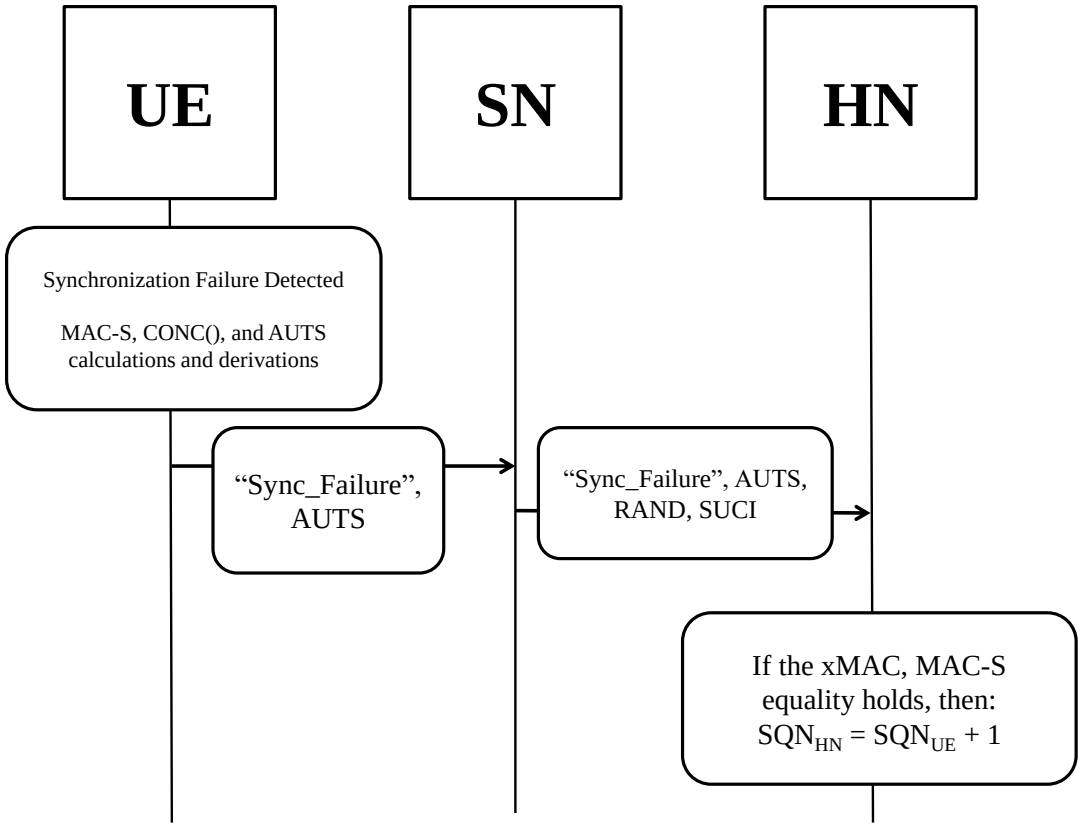

Figure 2.18: 5G-AKA Synchronization Failure Handling, from (pp. 1387) [24]

2.3.5 5G-AKA UE-Side Confirmation Solutions

The original 5G-AKA protocol described by the 3GPP has a shortcoming: the UE-entity is not explicitly notified of a successful or unsuccessful authentication (pp. 1393) [24]. Although the user would be aware of a successful authentication once their network connection starts working after step 12, there is no explicit step in the 5G-AKA protocol to convey this message. According to Basin et al., 5G-AKA “...only requires *implicit* authentication properties for the subscribers” (pp. 1393) [24].

There are two proposed solutions to this issue of key confirmation. This Section 2.3.5 discusses the respective protocol changes that each solution proposes. The two proposed solutions are cited from [24]. We included both solutions as part of our 5G-AKA imple-

mentation. They are selected using a Boolean flag within our Java code.

2.3.5.1 UE-Side Confirmation Solution 1

The first solution is to bind the *AUTN* value to the Serving Network Name (SNname) value (pp. 1394) [24]. This serves as proof of the HN committing to a specific SNname value (pp. 1394) [24], and the subscriber is able to perform verification of "...the authenticity of the challenge that commits to a specific SNname" (pp. 1394) [24].

The original structure for the AUTN and MAC (pp. 1394) [24] values are as follows:

$$\text{AUTN} := (\text{SQN} \oplus \text{AK}) \| \text{AMF} \| \text{MAC}$$
$$\text{MAC} = f1(K, \langle \text{SQN}_{\text{HN}}, \text{R} \rangle)$$

This proposed solution would redefine the MAC derivation as follows (pp. 1394) [24]:

$$\text{MAC} = f1(K, \langle \text{SQN}_{\text{HN}}, \text{R}, \text{SNname} \rangle)$$

This first proposed solution requires changes to Step 1 and Step 7 of the 5G-AKA protocol. We discuss how the authentication procedure would differ from the original version proposed by 3GPP documentation in these specific steps.

Changes to Step 1. The MAC value derivation function *f1* now includes the SNname value. This consequently changes the AUTN value, and ultimately the 5G HE AV is changed.

Changes to Step 7. This step is affected during the process of AV freshness verification. A new value (XMAC) is calculated at the UE using values received from the SN entity. The XMAC is compared against the original MAC value derived at the HN, to see if XMAC = MAC. Under the first proposed solution by Basin et al. (pp. 1394) [24], the SNname is used to derive the MAC value. Therefore, the same SNname value must be used to derive the XMAC.

2.3.5.2 UE-Side Confirmation Solution 2

The second solution is to create an additional final (thirteenth) step to the 5G-AKA protocol. This additional step is a unidirectional message sent from the SN to the UE (pp. 1394) [24]. The message is "...MACed with a key derived from K_{SEAF}, sent by the SN to the subscribers, at the very end of the protocol" (pp. 1394) [24]. In our implementation of this second solution, the HMAC-SHA-256 function is used for these final MAC implementations at both the SN and the UE. This second solution is shown in Figure 2.19. We discuss the components of the proposed thirteenth step below.

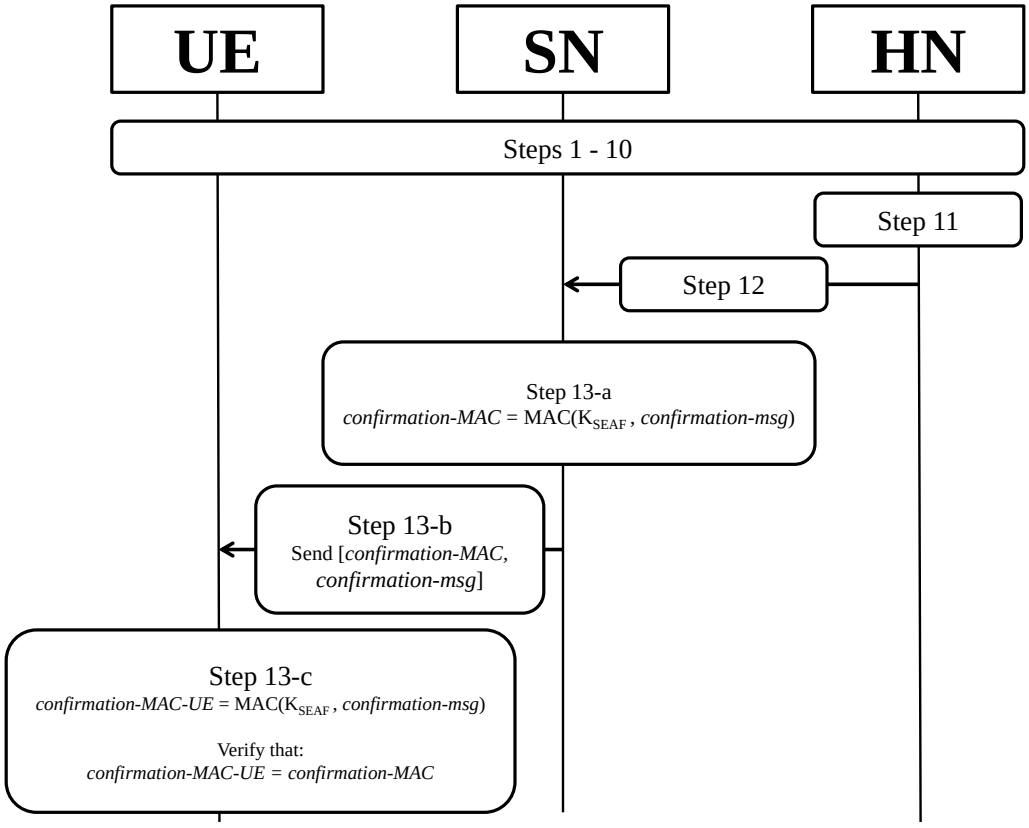

Figure 2.19: UE-side Confirmation Solution 2 UML Sequence Diagram

Step 13-a. The thirteenth step starts by generating a MAC tag using the K_{SEAF} key value and a confirmation message (string value) labelled *confirmation-msg*. The MAC tag

is labelled *confirmation-MAC*. The *confirmation-MAC* is generated at the SN.

Step 13-b. The *confirmation-msg* and *confirmation-MAC* are sent to the UE. In the Java implementation, these values are sent in a string value.

Step 13-c. The UE entity receives this final message, and extracts the *confirmation-msg* value. The UE then generates its own MAC tag using its local K_{SEAF} key value and the *confirmation-msg*. This new MAC tag is labelled *confirmation-MAC-UE*. If the *confirmation-MAC* and *confirmation-MAC-UE* values coincide, then the UE-side confirmation is successful. Therefore, the subscriber is clearly notified that the 5G-AKA process is successful, and that the HN is authenticated.

2.4 Open-Source Project

The ultimate purpose of our 5G-AKA implementation is to be published online as an open-source project. An implementation of this protocol would be valuable as a template. Researchers with criticisms of the existing 5G-AKA protocol structure could easily download and examine our implementation. They could also tinker with our implementation in order to identify potential points of weakness in the protocol. Such open availability could result in security improvements for the 5G-AKA protocol from researchers and practitioners.

These goals would require the open-source implementation to be published under a license that does not restrict the access and development permissions of the 5G-AKA protocol code. We considered different licenses in order to select the best option.

First, we considered using the Apache License (Version 2.0). With this license, each individual or entity that makes a contribution to a software product is granted "...a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form" [49]. Contributors are able to to "...reproduce and distribute..." [49] software under the Apache License, but there are some conditions. Contributors must include a copy of the license when sharing the software [49], and ensure that edited files contain "prominent notices" [49] explaining that they changed the files. All previously existing copyrights, patents, trademarks, etc. must also be included in new derivative versions of the software [49].

We also considered using the GNU General Public License. In overview, this license allows developers to assert copyright on their software and then grant permissions to the public to "...copy, distribute, and/or modify" [50]. We found this license agreement to have a very extensive list of terms and conditions [50]. We felt this license may be too complex for our 5G-AKA proof-of-concept.

Lastly, we considered the MIT license. The MIT license allows software and associated documentation to be available free-of-charge on an online platform [51]. Under this license, anyone would be allowed to "...use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies..." [51] of the 5G-AKA protocol implementation. The purpose of the original MIT license was to allow developers to release software for free "...with a copyright

notice that identified where it came from and did not require any signatures" (pp. 95) [52]. This would allow useful software to have the widest possible influence on whatever technological fields it pertained to (pp. 96) [52]. Publishing the 5G-AKA protocol under the MIT license could allow it to have a potentially wider influence on 5G network authentication, and encourage the development of new features. Out of the three licenses discussed, we found the MIT license to be the simplest and least restrictive.

Based on our analysis of license options, we decided that our implementation of the 5G-AKA protocol will be published under the MIT license.

We intend to release our implementation code online on a public Git repository. The repository will include a brief explanation of the repository's contents. It will also include instructions on how to run our implementation.

Chapter 3

Usability Analysis of OpenLDAP Access Control Lists (ACLs)

In this chapter, we discuss the usability analysis of OpenLDAP ACLs using error categorization. In Section 3.1, we explain the purpose of the OpenLDAP protocol and its syntactic structure. In Section 3.2, we summarize the prior work that has been completed in order to assess the usability of OpenLDAP. This includes a cognitive walkthrough to assess the protocol. In this section we discuss also the human-subject study that was used to collect usability data. We discuss the structure of the study, and the sample policy that each participant was required to implement. In Section 3.3, we discuss the analysis of the collected usability data.

We used HTA (Hierarchical Task Analysis) diagrams in order to break down the study's sample policy into sub-goals. This would allow us to better pinpoint sources of errors made by participants, and categorize these errors. In Section 3.4, we explain the two Hierarchical Task Analysis approaches that we used for error categorization, and the four error categories. Lastly, in Section 3.5, we discuss the analysis of the error categorization data.

It should be noted that the citation labelled [26] pertains to a paper that I co-authored. The content of this paper has been cited extensively in this chapter.

3.1 Background

The term LDAP stands for "Lightweight Directory Access Protocol" (pp. 3) [53]. OpenLDAP is designed to provide users with "directory services" (pp. 3) [53]. It specifically pertains to directory services based upon X.500 (pp. 3) [53]. A directory is defined as "...a specialized database specifically designed for searching and browsing, in additional to supporting basic lookup and update functions" (pp. 3) [53]. Therefore, directory services are "...used to store a variety of data such as employee information and passwords, and can be seen as a critical infrastructure component of an enterprise" (pp. 1) [26].

Essentially, OpenLDAP can be used to manage the access privileges (e.g. read) that different users may possess for different documents within a directory. The OpenLDAP Access Control List (ACL) controls such privileges within a Directory Information Tree (DIT) (pp. 2) [26]. The ACL is made up of access directives that have the following syntactical structure (pp. 2) [26]:

```
<access directive> ::= to <what> [by <who> [<access>] [<control>] ]+
```

The <what>, <who>, <access>, and <control> components of the directive are called clauses (pp. 2) [26]. Table 3.1 summarizes the significance of each clause.

Clause	Specification
<what>	Resource to which access is granted
<who>	User to whom access is granted
<access>	Type of access granted
<control>	Outlines allotment or denial of access request

Table 3.1: Summary of OpenLDAP Access Directive Clauses, from (pp. 2) [26]

Another important concept pertaining to OpenLDAP ACLs is the precedence rule. This rule is used to determine whether or not an access request is granted by the ACL (pp. 2) [26]. The precedence rule is described as "complex" (pp. 12) [25], and it is applied on a "...case by case basis..." (pp. 12) [25]. A case, in this context, consists of a single resource and a single principal (pp. 12) [25]. When there is some conflict of principals and resources within directives, "...the precedence rule is used to determine which directive takes effect" (pp. 12) [25].

Let us assume that the following access request exists within an ACL (pp. 2) [26]:

$$q = \langle\ r,\ u,\ a\ \rangle$$

In this sample access request q, the r value represents the resource that user u is requesting for action a (pp. 2) [26]. Due to the precedence rule, each directive that matches the resource r is applied (pp. 2) [26]. The precedence rule then seeks to confirm if "...the <who> clause in the directive for which r matches the <what> clauses matches the user u in the request" (pp. 2) [26]. Lastly, the precedence rule must check the <access> clause (pp. 2) [26].

In order to assess the usability of OpenLDAP ACLs, a human-subject study was devised in [25]. The study trained users in OpenLDAP syntax, and then asked them to try and implement an access control system based upon a set of policy items.

In this study, usability is defined as "...the ease with which a human administrator can express a policy in an ACL..." (pp. 1) [26]. It has been concluded from the results of this study that OpenLDAP has poor usability. Graphs and data analysis have been used to identify potential reasons for poor usability [25] [26]. Data analysis has also been done to suggest potential improvements that would improve OpenLDAP usability [25] [26].

The focus of the remainder of this chapter consists of two main parts. First, we summarize the prior work completed in analyzing the data in order to explain why OpenLDAP has such poor usability. We discuss this in Sections 3.2 and 3.3. Second, we discuss our categorization of errors that we observed in the participants' data. We discuss this in Sections 3.4 and 3.5.

3.2 Cognitive Walkthrough and Human-Subject Study

Before the study with human-participants, a cognitive walkthrough was completed to assess the usability of the OpenLDAP system (pp. 3) [26]. This entailed "...a person performing tasks that a typical user of the system would perform and assessing difficulties" (pp. 3) [26]. This cognitive walkthrough was completed by a participant who posessed expertise in access control, but no expertise in LDAP or OpenLDAP ACLs (pp. 3) [26]. One outcome of this walkthrough was that this participant required a considerable amount of extra time to self-train based on provided documentation (pp. 3) [26]. This participant also struggled with the complex syntax and policy-order rules for ACL implementations using OpenLDAP (pp. 3) [26]. These initial findings indicate that OpenLDAP has poor usability. A study using human participants was designed based on the findings from the cognitive walkthrough. This study would be used to continue the investigation of OpenLDAP usability.

The structure of the human-subject study consists of the following components for each session: a period of training, followed by a task of policy implementation (pp. 5) [26].

It was established that an OpenLDAP ACL is "...configured by systems administrators in enterprise settings rather than end-users of personal computers and devices" (pp. 5) [26]. Consequently, the study recruited "...senior-year and graduate students in computer science and engineering" (pp. 5) [26]. The study required all participants to have a working knowledge of UNIX (or Linux) command-line shell (e.g. bash) (pp. 5) [26]. The study also required that participants did not have any prior familiarity or knowledge of LDAP systems (pp. 5) [26]. These requirements were an attempt to ensure that all participants in the study would possess the technical skills of a systems administrator, with a level playing field in terms of knowledge of LDAP (pp. 5) [26]. Each participant signed a consent letter allowing their session to be recorded through screen capture software and audio recording (pp. 27-28) [25].

The study trained voluntary participants on the basics of OpenLDAP syntax rules. The training sessions were 30 minutes in length (pp. 6) [26]. Each participant was provided with a sample syntax sheet that could be used as a reference during implementation (pp. 6) [26]. The training was done through simplified documentation and sample OpenLDAP implementations (pp. 21) [25]. Participants were shown six example ACL implementations (pp. 6) [26] that all tried to achieve the same hypothetical policy (pp. 22) [25]:

1. Alice has write permission to herself
2. Bob has write permission to himself
3. Alice and Bob have read permissions to each other

The diagram in Figure 3.1 is a DIT that illustrates this sample policy.

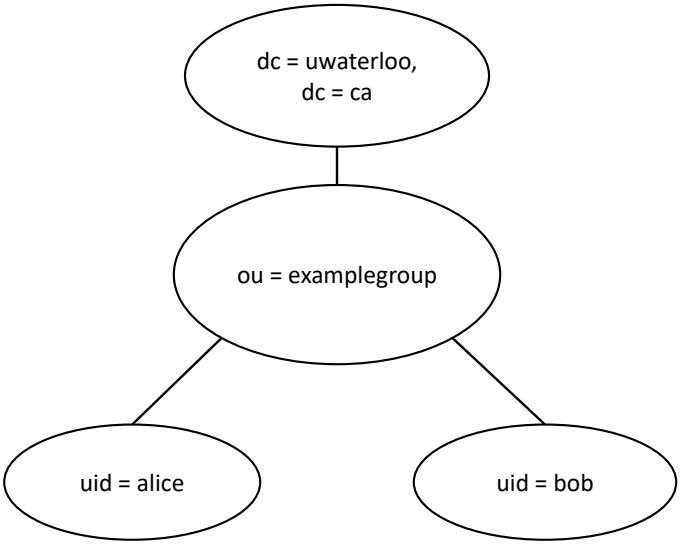

Figure 3.1: Directory Information Tree of Sample Policy, from (pp. 22) [25]

Each of the six example implementations were shown to every participant (pp. 6) [26]. Explanations were given as to why each one was correct or incorrect in terms of their respective OpenLDAP syntax (pp. 6) [26].

Once a user was given their introductory OpenLDAP training session, they were then asked to implement a high-level policy on a computer with OpenLDAP software installed (pp. 17) [25]. This policy consists of 7 goals (pp. 17) [25]:

1. admin has write permission to everything in the tree
2. every user has write permission to her/his own attributes (displayName, mobile, telephoneNumber, and userPassword)

3. humanresources has write permission to all employee entries
4. sambaservice has read permission to all userPasswords
5. managers of engineering and accounting have write permission to all their users
6. anonymous has authentication permission to all userPasswords
7. all employees can read the displayName, mobile and telephoneNumber attributes of all other employees

This policy is based upon an example from a blog by Ingo Bente [54]. The design of this policy implementation task was to describe a "...complete policy for a DIT..." (pp. 5) [26] in an "itemized" (pp. 5) [26] fashion. This would allow participants to start by implementing each policy goal individually before editing their overall ACL implementation (pp. 5) [26].

Each participant session was limited to being 1 hour in length (including the 30-minute training session) (pp. 6) [26]. Participants were also limited to using a "...restricted syntax of OpenLDAP ACLs only" (pp. 6) [26]. This was done to mitigate the issue of OpenLDAP ACLs having a very complex and extensive syntax (pp. 5) [26].

Upon completion of their attempted solution, the elapsed time for each participant was noted (pp. 30) [25]. The participant's confidence in their implementation's correctness was also noted (pp. 31) [25]. In total, the usable collected data consisted of implementations by 48 individual participants (pp. 30) [25]. Each participant's implementation was checked for correctness, and failed goals were noted.

3.3 Human-Subject Study Data Analysis

It was found that the average time-to-completion was 25:44 minutes across the 48 participants (pp. 7) [26]. The standard deviation was 6:50 minutes (pp. 7) [26]. The average level of self-reported confidence by participants (with respect to implementation correctness) was approximately 71.1% (pp. 7) [26]. It was determined that "...the point biserial correlation between self-reported correctness and actual correctness was 0.0275 only" (pp.

7) [26]. This could be interpreted as participants being overly confident in their OpenLDAP implementation abilities (pp. 7) [26].

The graphs shown in Figure 3.2 and Figure 3.3 visualize the goal-based nature of the study's policy that was implemented by the study's participants. The graph in Figure 3.2 shows the number of participants who completed each of the seven goals successfully. This graph includes an extra category of "no extra permissions", signifying participants that did not award additional permissions that were not explicitly specified in the sample policy (pp. 7) [26]. It is clear from this graph that goals 4 and 5 had the largest amount of failed implementations across all participants. Goal 4 requires the following: *sambaservice has read permission to all userPasswords.* Goal 5 requires the following: *managers of engineering and accounting have write permission to all their users.*

The graph in Figure 3.3 shows the total number of participants who completed different amounts of policy goals successfully in their respective implementations. This graph includes a category of "7 + no extra permissions", signifying participants that accomplished all 7 goals without implementing any extra unspecified permissions (pp. 7) [26]. It is clear that 18 out of 48 participants were only able to successfully accomplish less than half of the 7 policy goals. In total, only 10 participants were able to complete all 7 policy goals without adding any extra permissions that were not explicitly defined within the ACL policy written goals (pp. 7) [26]. These statistics support the idea that OpenLDAP has poor usability.

One potential reason for the higher incidence of failure for goal 4 could be that this goal "...is associated with the same resource..." (pp. 7) [26] as goal 6. Goal 6 requires the following: *anonymous has authentication permission to all userPasswords.* Participants may have had trouble understanding how to implement two separate policy goals that both require permissions to the same "userPasswords" attribute.

One potential reason for the higher incidence of failure for goal 5 could be that this goal has a similar wording in its instructions as goal 3. Goal 3 requires the following: *human-resources has write permission to all employee entries.* Both goals 3 and 5 describe users that would have write permissions to other user entries. This may have been a source of confusion for participants. Another potential reason could be that goal 5 affords its write permissions strictly to managers in the engineering and accounting sub-trees. Participants may have had difficulty discerning managers from regular employees for each respective sub-tree.

In the collected data, it was observed that there were many different types of errors made by participants. The next section of this thesis deals with the categorization of these errors.

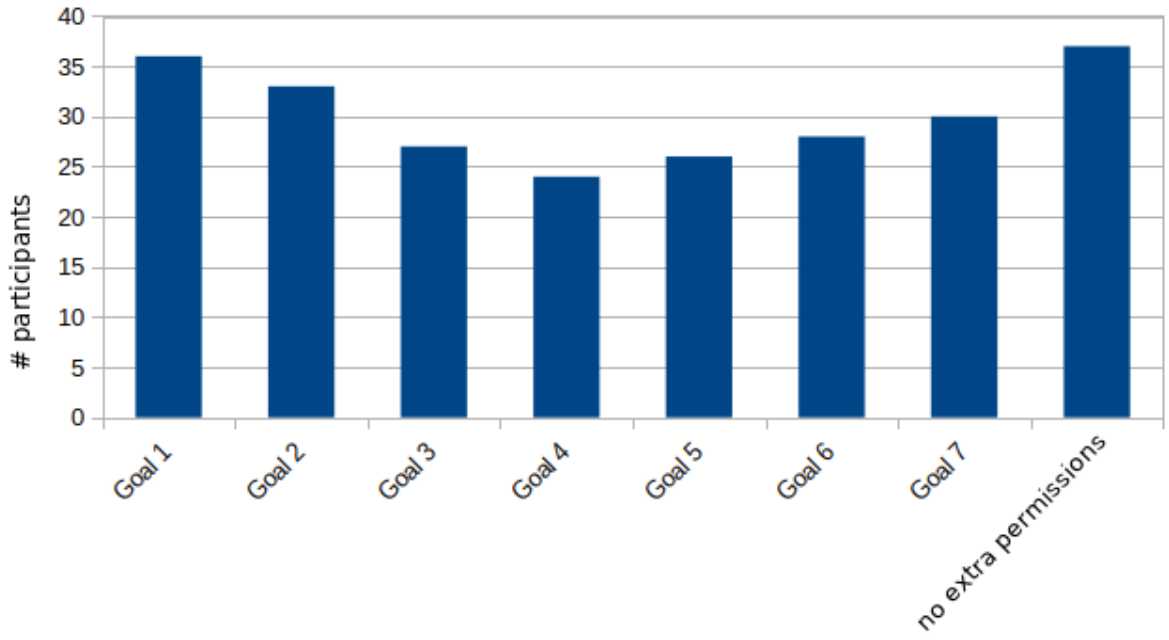

Figure 3.2: Number of Participants Who Completed Goal Successfully vs. Goal Number, from (pp. 8) [26]. Total of 48 participants.

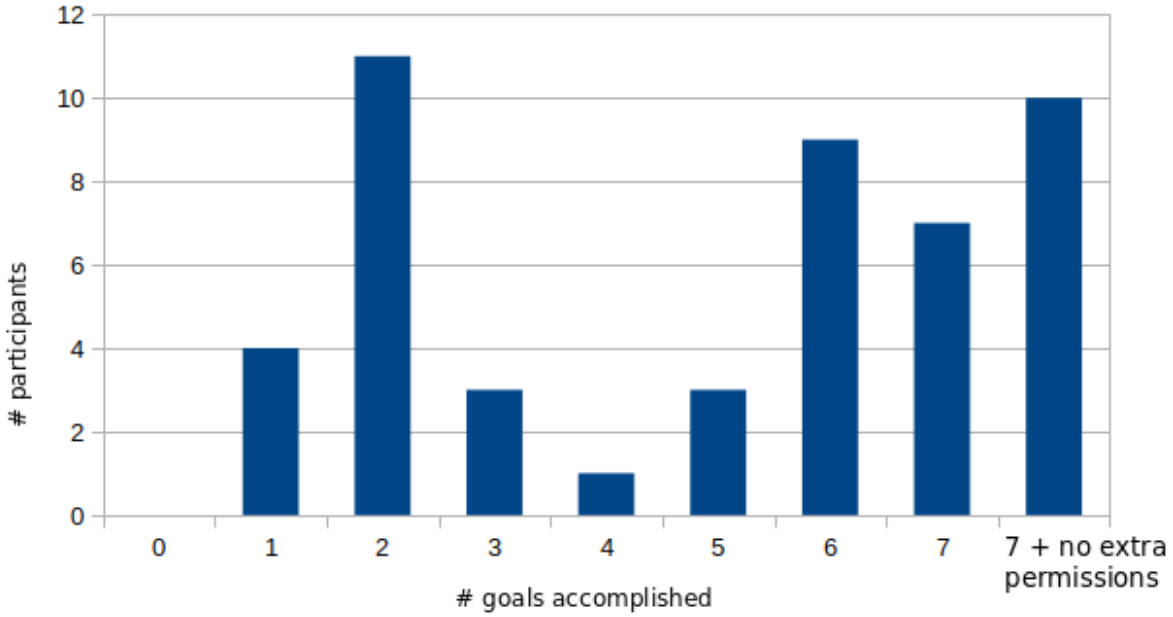

Figure 3.3: Number of Participants vs. Total Number of Goals Accomplished, from (pp. 8) [26]. Total of 48 participants.

3.4 Error Categorization Approaches and Categories

For data analysis, prior work was focused on identifying how often each of the 7 goals were implemented correctly across submissions (pp. 30-32) [25]. It also focused on how many goals were successfully implemented per each submission (pp. 30-32) [25]. This was done to suggest potential reasons and rationalizations for why certain goals were implemented incorrectly as compared to others (pp. 32-34) [25]. This sort of analysis helps to explain why specific goals failed, but it does not clearly identify the types of errors that are observed in goal implementations. Such data analysis requires the creation of error categories.

The basis for our error categorization is the THEA technique (pp. 1) [33]. THEA is a systematic method to identify errors and understand their consequences in conjunction with contextual factors (e.g. environmental factors) (pp. 2) [33]. THEA error analysis utilizes "cognitive failure" (pp. 15) [34] types that describe "...ways in which human information processing can fail, which have the potential for leading to erroneous behaviour" (pp. 15) [34].

In order to correctly model goals within a system, HTA diagrams are used. These diagrams break system goals down into smaller sub-goals that must all be satisfied such that the overall goal is achieved (pp. 3) [33]. Example HTAs depicted in (pp. 10, 23) [34] provide a basis for our own HTA development.

We use two separate approaches to create the HTA diagrams. In Approach-1, we create seven HTA diagrams, one for each goal of the study's policy. In Approach-2, we create a single overall HTA diagram that covers all seven goals.

In this section, we discuss Approach-1 and Approach-2 in detail. We provide explanations for each HTA diagram. We discuss also the four THEA error categories that are used to classify the errors made by each participant in their respective policy implementation during their session.

3.4.1 Approach-1

The participant study required the implementation of a policy consisting of seven goals. Approach-1 uses seven HTA diagrams for modelling purposes.

Each HTA diagram pertains to a respective policy goal. These goal-specific HTA diagrams summarize the steps necessary to achieve each of the seven policy goals, and in what order these steps must be completed. The HTA diagrams for goal 2 and goal 5 were also included in (pp. 8) [26].

3.4.1.1 Goal 1 HTA Diagram

The HTA diagram in Figure 3.4 pertains to the OpenLDAP policy goal 1: *Admin has write permission to everything in the tree.*

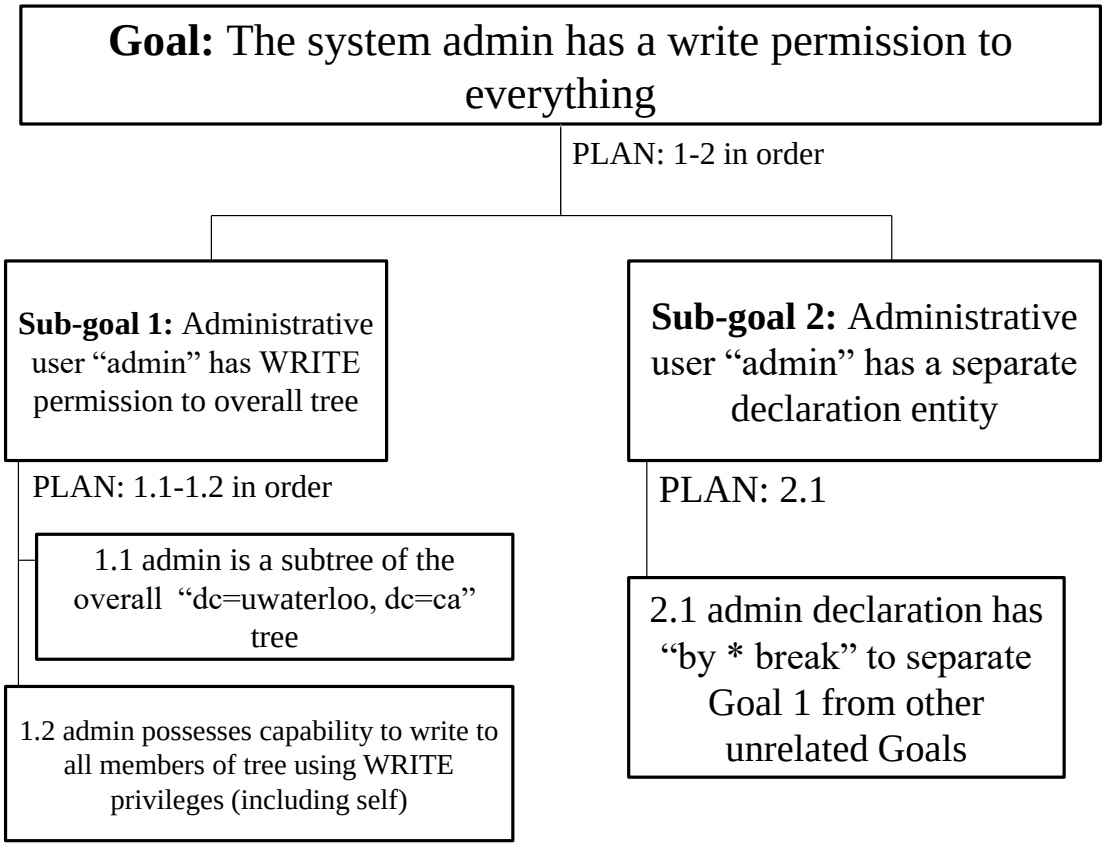

Figure 3.4: Goal 1 HTA Diagram

This goal is broken down into two main sub-goals to be completed in numerical order:

1. Administrative user "admin" has WRITE permission to the overall tree
2. Administrative user "admin" has a separate declaration entity

Sub-goal 1 requires two in-numerical-order steps. First, the admin would need to be declared as the subtree of the overall policy tree. Second, the admin must be given writing

privileges to all tree members (including themselves).

Sub-goal 2 simply requires an explicit declaration separator to keep it separate from the implementations of the remaining 6 goals. This would be done using the OpenLDAP syntax “by * break”.

3.4.1.2 Goal 2 HTA Diagram

The HTA diagram in Figure 3.5 pertains to the OpenLDAP policy goal 2: *Every user has write permission to her/his own attributes (displayName, mobile, telephoneNumber, and userPassword).*

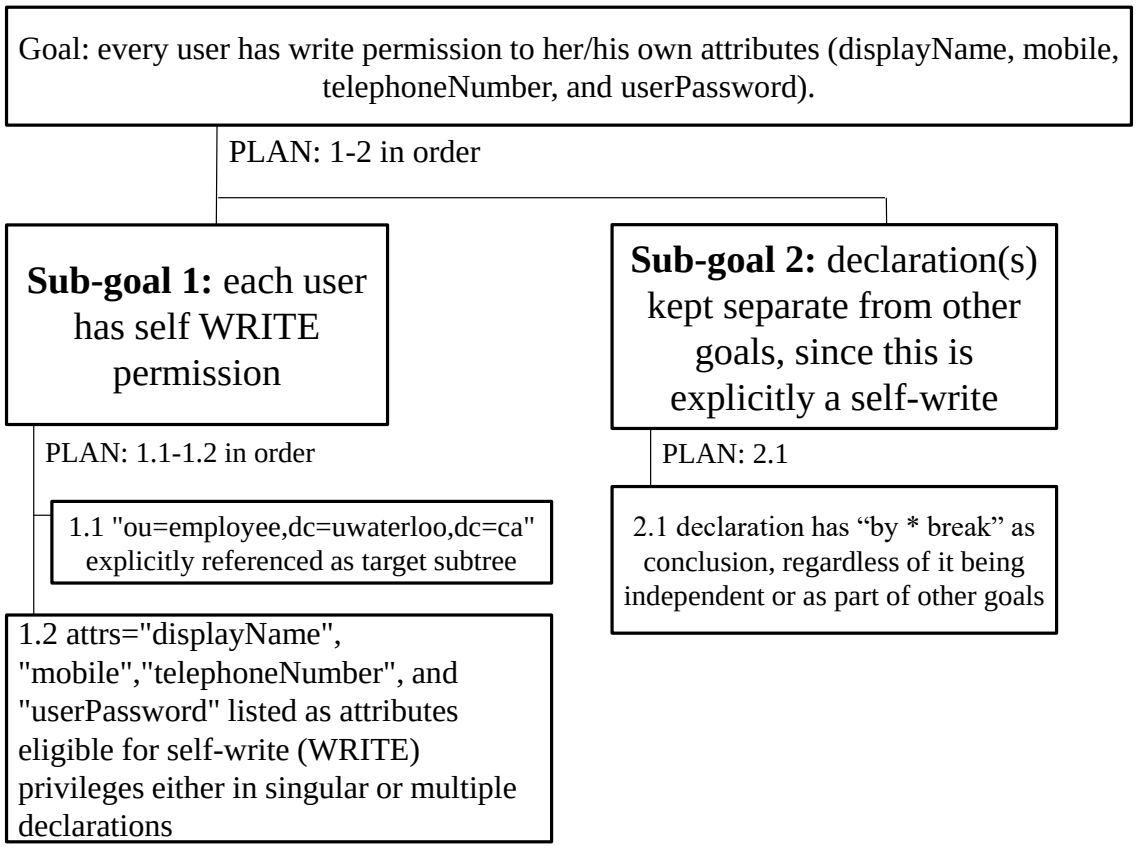

Figure 3.5: Goal 2 HTA Diagram

This goal is broken down into two main sub-goals to be completed in numerical order:

1. Each user has a self-writing permission
2. These declarations are kept separate from other goals

Sub-goal 1 requires two in-order steps. First, the overall tree is referenced as the target subtree for the goal. Second, the implementation must list the explicit attributes whose respective users have self-write privileges over.

Sub-goal 2 simply requires an explicit declaration separator to keep it separate from the implementations of the remaining 6 goals. This would be done using the OpenLDAP syntax “by * break”.

3.4.1.3 Goal 3 HTA Diagram

The HTA diagram in Figure 3.6 pertains to the OpenLDAP policy goal 3: *humanresources has write permission to all employee entries.*

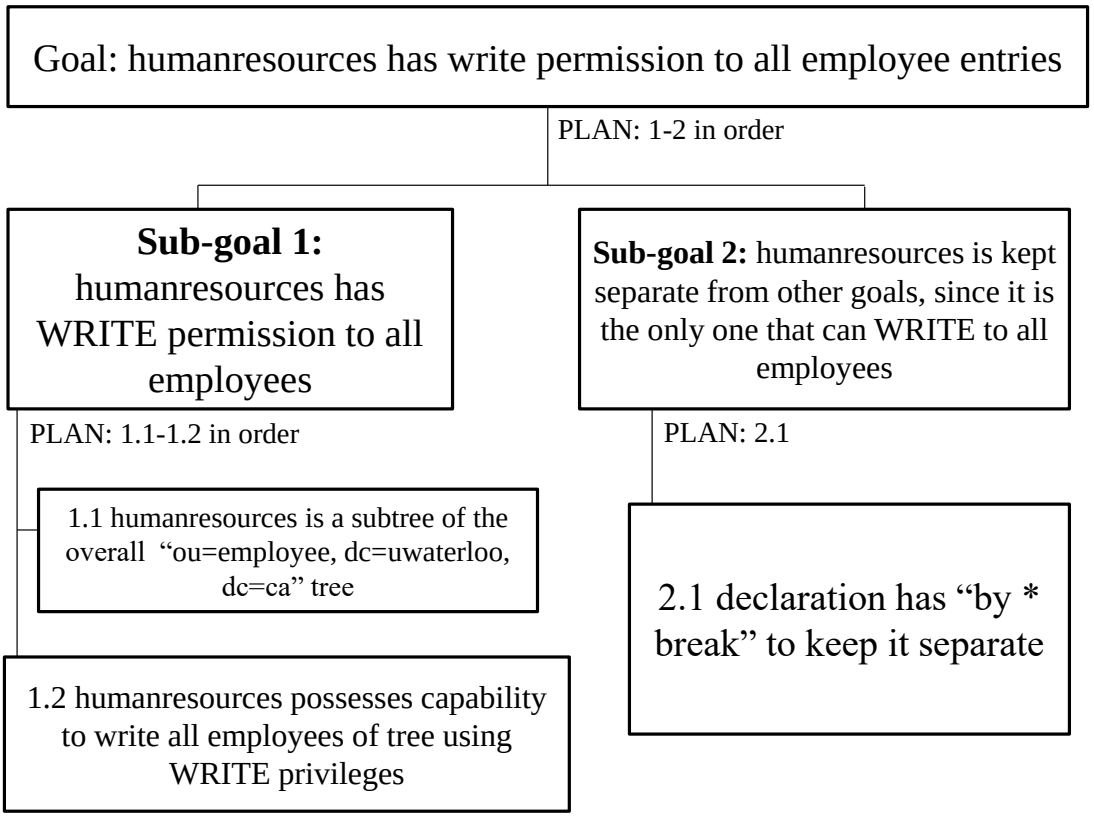

Figure 3.6: Goal 3 HTA Diagram

This goal is broken down into two main sub-goals to be completed in numerical order:

1. "humanresources" has WRITE permission to all employees
2. "humanresources" is kept separate from other goals, since it is the only one that can WRITE to all employees

Sub-goal 1 requires two in-order steps. First, the "humanresources" user is a sub-tree of the overall employee sub-tree. Second, the "humanresources" user is given writing-privileges to all users designated as "employees" within the tree.

Sub-goal 2 simply requires an explicit declaration separator to keep it separate from the implementations of the remaining 6 goals. This would be done using the OpenLDAP syntax "by * break".

3.4.1.4 Goal 4 HTA Diagram

The HTA diagram in Figure 3.7 pertains to the OpenLDAP policy goal 4: *sambaservice has read permission to all userPasswords.*

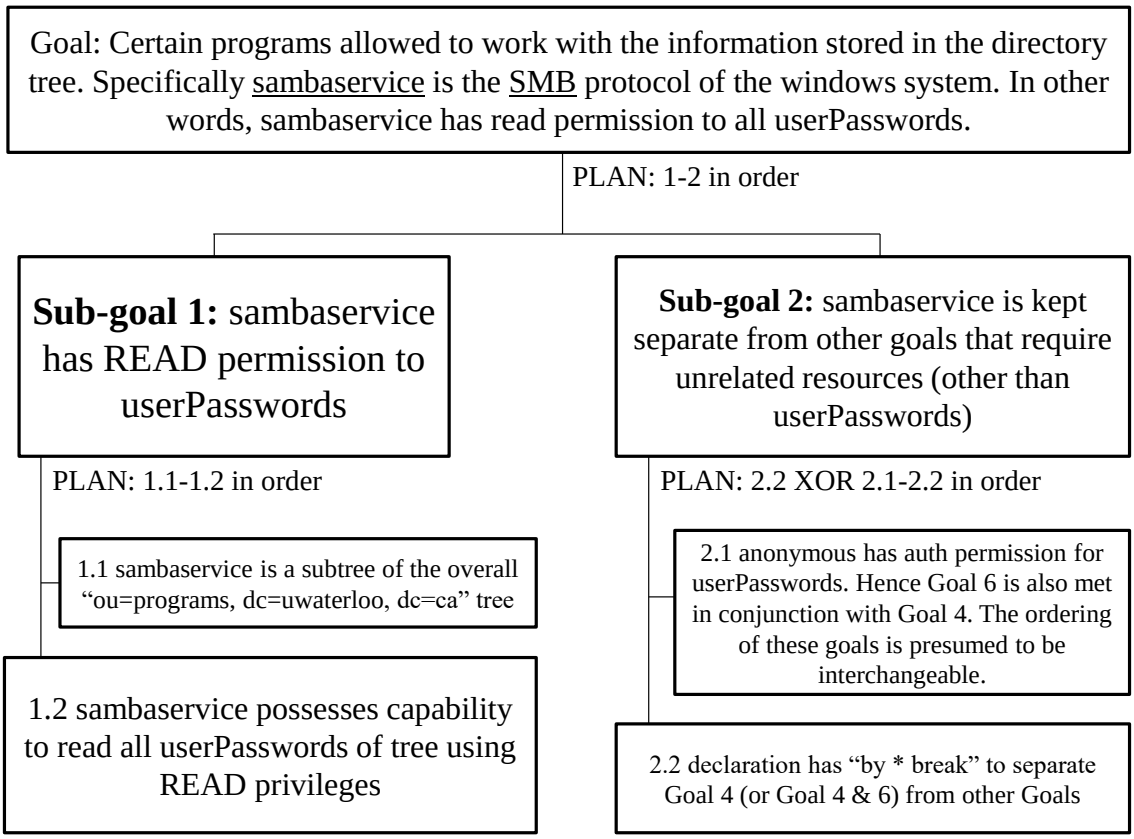

Figure 3.7: Goal 4 HTA Diagram

This goal is broken down into two main sub-goals to be completed in numerical order:

1. The “sambaservice” user has read-privileges to userPasswords
2. The “sambaservice” user is kept separate from other goals that need resources other than userPasswords

Sub-goal 1 requires two in-order steps. First, “sambaservice” is a sub-tree of the overall programs tree. Second, “sambaservice” is given read-privileges to all userPasswords for users within the tree.

Sub-goal 2 includes two steps. In step 2.1, “anonymous” can possess auth. permission for userPasswords, completing both goals 4 and 6. This step is not integral to achieving goal 4. It is an additional step that can achieve both goals 4 and 6 in a single policy goal implementation.

In step 2.2, the goal needs an explicit declaration separator to keep it separate from the implementations of the remaining 6 goals. This would be done using the OpenLDAP syntax “by * break”. As shown in Figure 3.7, step 2.1 is optional for goal 4 to be achieved. This sub-goal can be achieved by implementing step 2.2 alone, or steps 2.1 and 2.2 in numerical order.

3.4.1.5 Goal 5 HTA Diagram

The HTA diagram in Figure 3.8 pertains to the OpenLDAP policy goal 5: *managers of engineering and accounting have write permission to all their users.*

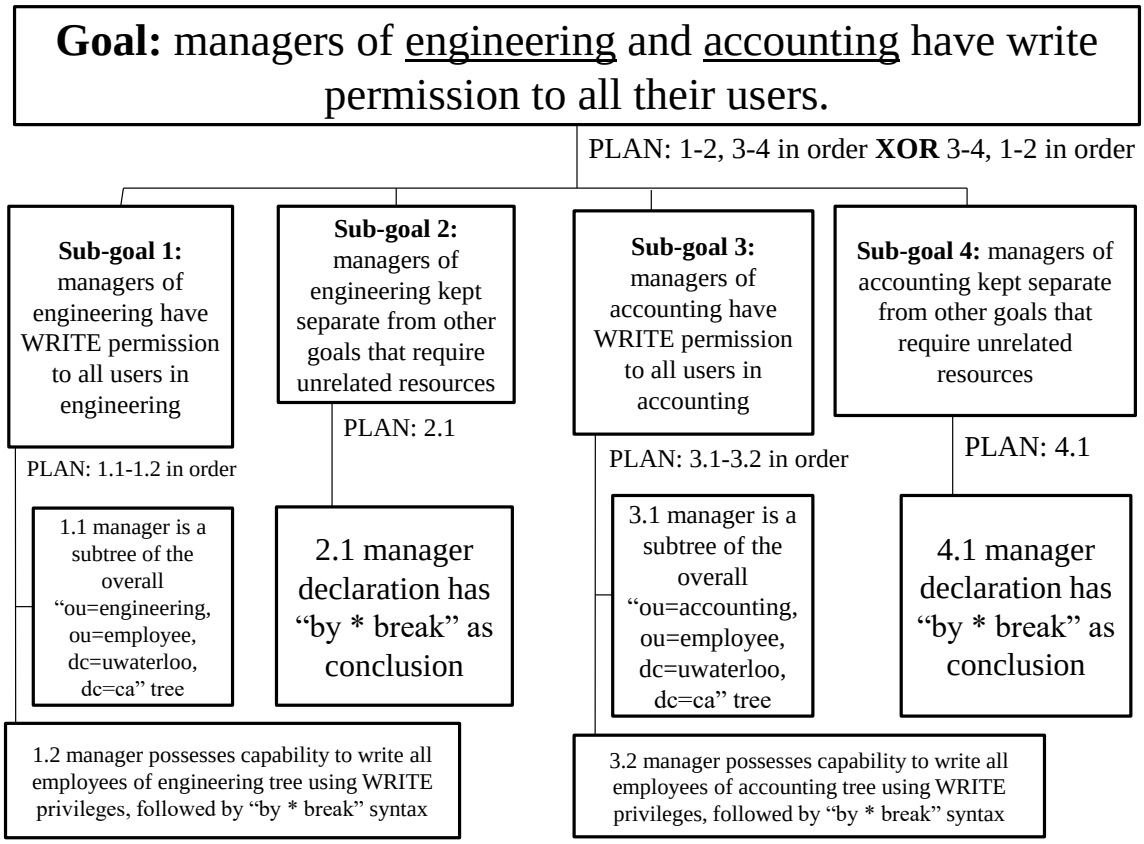

Figure 3.8: Goal 5 HTA Diagram

This goal is broken down into four main sub-goals:

1. Managers of engineering have WRITE permission to all users in engineering
2. Managers of engineering are kept separate from other goals that require unrelated resources
3. Managers of accounting have WRITE permission to all users in accounting
4. Managers of accounting are kept separate from other goals that require unrelated resources

These sub-goals can be completed in one of two orderings. The first option would be to complete sub-goals 1 to 4 in numerical order. The alternative option would be to complete sub-goals 3 and 4 (in numerical order), followed by completing sub-goals 1 and 2 (in numerical order). As shown in figure 3.8, these two options exist as an exclusive-or selection.

Sub-goal 1 requires two in-order steps. First, "manager" is a sub-tree of the "engineering" sub-tree. Second, "manager" is given write-privileges to all employees for users within the "engineering" sub-tree. This second part requires an explicit declaration separator using the OpenLDAP syntax "by * break".

Sub-goal 2 requires a single step. The engineering manager declaration has a "by * break" syntax conclusion.

Sub-goal 3 requires two in-order steps. First, a new "manager" is a sub-tree of the "accounting" sub-tree. Second, "manager" is given write-privileges to all employees for users within the "accounting" sub-tree. This second part requires an explicit declaration separator using the OpenLDAP syntax "by * break".

Sub-goal 4 requires a single step. The accounting manager declaration has a "by * break" syntax conclusion.

It should be noted that there is some overlap between steps 1.2 and 2.1. Both require a "by * break" separation. However, step 2.1 explicitly justifies why this statement is required for the overall goal implementation. This overlap also exists between steps 3.2

and 4.1. A future iteration of this diagram could simply remove the mention of a "by * break" in steps 1.2 and 3.2.

3.4.1.6 Goal 6 HTA Diagram

The HTA diagram in Figure 3.9 pertains to the OpenLDAP policy goal 6: *anonymous has authentication permission to all userPasswords.*

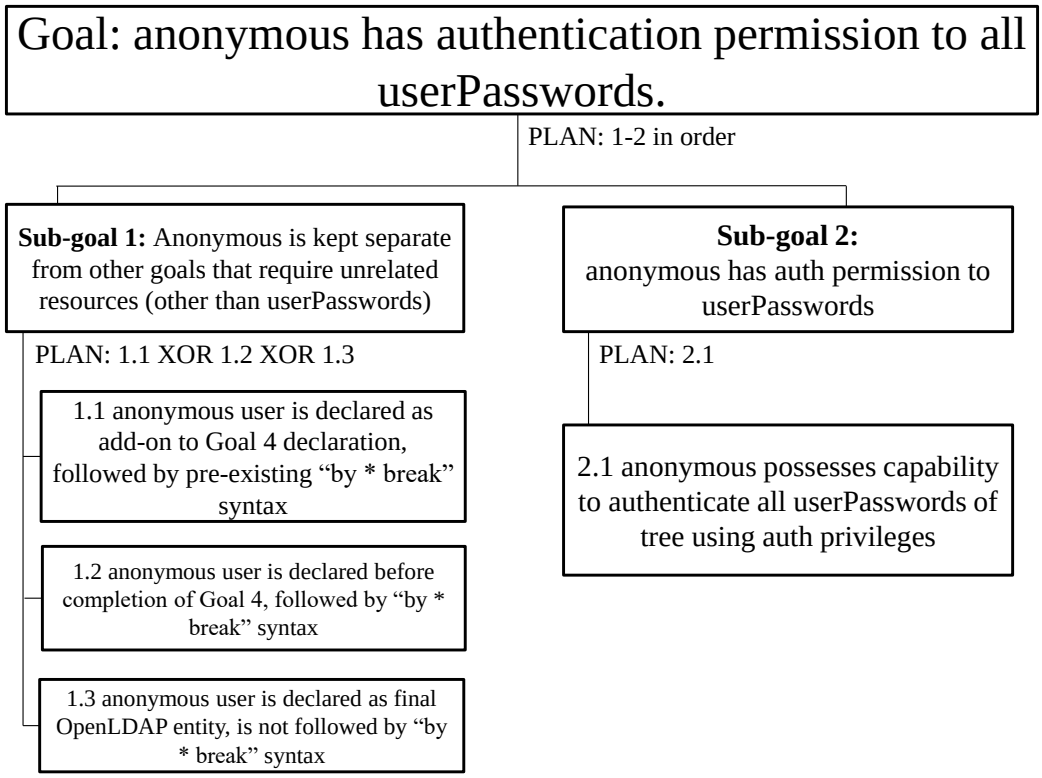

Figure 3.9: Goal 6 HTA Diagram

This goal is broken down into two main sub-goals to be completed in numerical order:

1. The "anonymous" user is kept separate from other goals that require unrelated resources (other than userPasswords)
2. The "anonymous" user has auth permission to userPasswords

Sub-goal 1 offers three potential steps. Only one of these steps is to be accomplished in order for sub-goal 1 to be considered complete. This is shown by the exclusive-or operator in Figure 3.9. In step 1.1, the "anonymous" user is declared as part of the existing goal 4 implementation. We discuss this earlier in this section as part of the Goal 4 HTA. In step 1.2, the "anonymous" user is declared before the implementation of goal 4. In step 1.3, the "anonymous" user is declared as the final OpenLDAP entity in the policy implementation. This third option would require the absence of the "by * break" separator syntax. One of these three options would fulfill the sub-goal 1.

Sub-goal 2 requires a single step. The "anonymous" user can authenticate all userPasswords using auth. privileges.

3.4.1.7 Goal 7 HTA Diagram

The HTA diagram in Figure 3.10 pertains to the OpenLDAP policy goal 7: *all employees can read the displayName, mobile and telephoneNumber attributes of all other employees.*

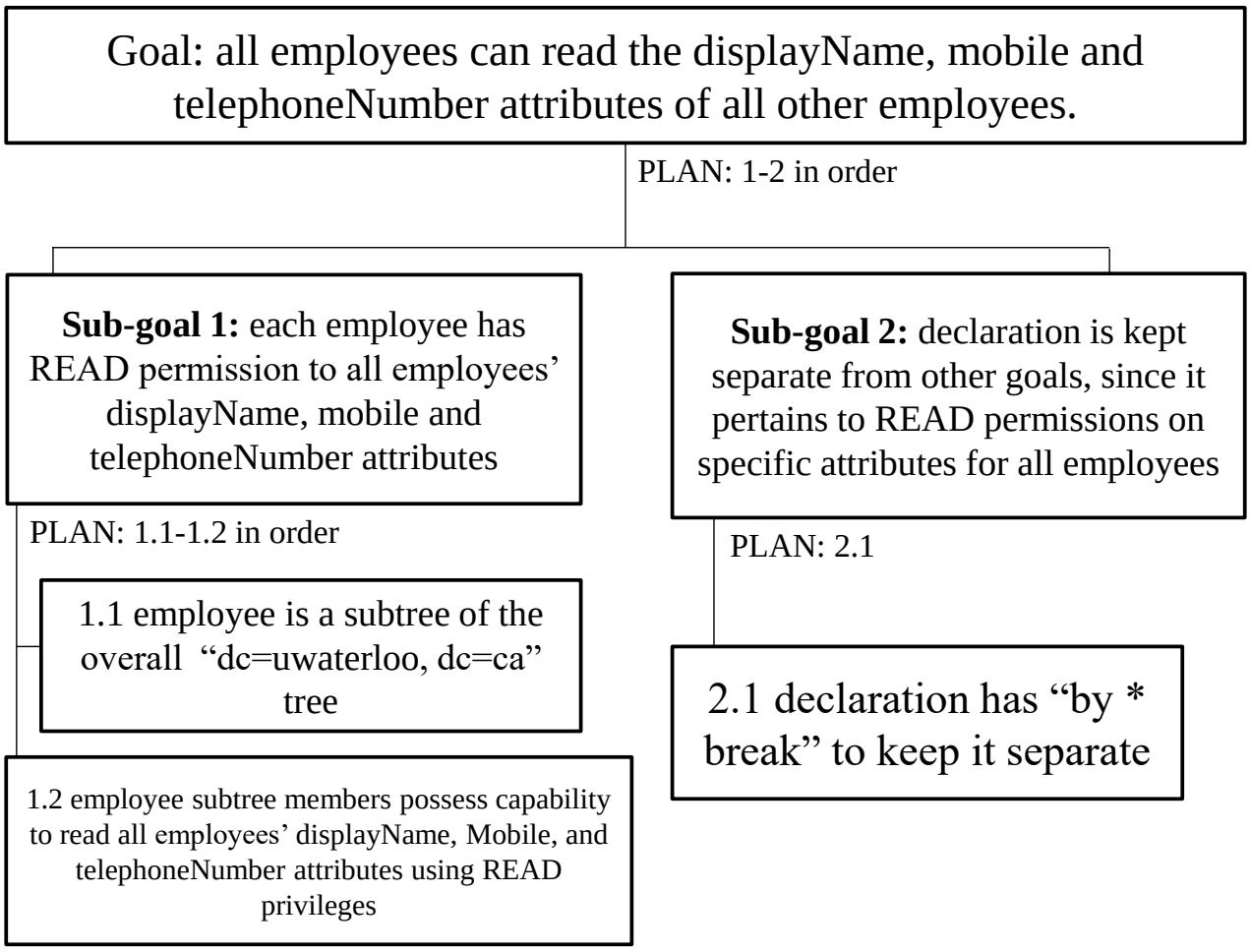

Figure 3.10: Goal 7 HTA Diagram

This goal is broken down into two main sub-goals to be completed in numerical order:

1. Each employee has READ permission to all employees' displayName, mobile and telephoneNumber attributes

2. Declaration is kept separate from other goals, since it pertains to READ permissions on specific attributes for all employees

Sub-goal 1 requires two steps to be completed in-order. First, the "employee" users are declared as a sub-tree for the overall policy tree. Second, each "employee" user possesses read-privileges for the displayName, Mobile, and telephoneNumber attributes of all other "employee" users.

Sub-goal 2 requires a single step. The declaration must have a "by * break" syntax conclusion.

3.4.2 Approach-2

An alternative to creating an HTA for each individual policy goal would be to assess the overall policy as a single system to be implemented. This would be done by using a single HTA diagram, assessing each user's implementation as an aggregate of all seven policy goals. We implement this "goal summary" (pp. 8) [26] method in our Approach-2 HTA.

This approach to error categorization delegates the policy's seven items as sub-goals. In Approach-2, the overall HTA goal is to implement an OpenLDAP policy that fulfills all of the specifications contained within the policy items.

3.4.2.1 Overall Goal-Summary HTA Diagram

The HTA diagram in Figure 3.11 illustrates the analysis method of Approach-2.

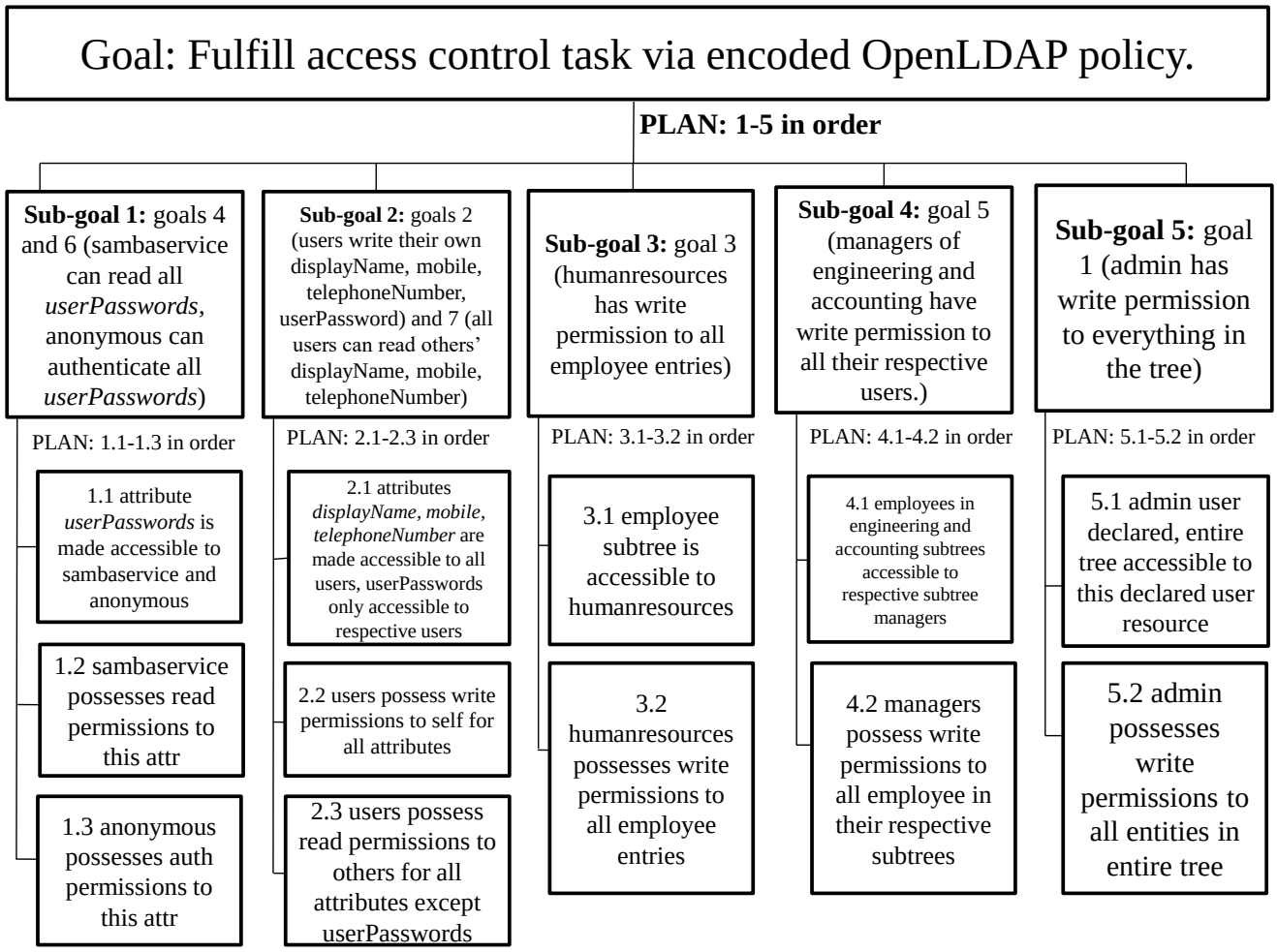

Figure 3.11: Overall Goal-Summary HTA Diagram

The goal in this HTA is broken down into 5 main sub-goals:

1. Goals 4 and 6 ("sambaservice" can read all userPasswords, "anonymous" can authenticate all userPasswords)
2. Goals 2 (users write their own displayName, mobile, telephoneNumber, userPassword) and 7 (all users can read others' displayName, mobile, telephoneNumber)
3. Goal 3 ("humanresources" has write permission to all employee entries)
4. Goal 5 (managers of engineering and accounting have write permission to all their respective users.)
5. Goal 1 ("admin" has write permission to everything in the tree)

Sub-goal 1 requires three steps to be completed in-order. First, the "userPasswords" attribute is made accessible to users within the "sambaservice" and "anonymous" sub-trees. Second, the "sambaservice" sub-tree possesses read-privileges to the "userPasswords" attribute. Third, the "anonymous" sub-tree possesses read-privileges to the "userPasswords" attribute.

Sub-goal 2 requires three steps to be completed in-order. First, the "displayName", "mobile", and "telephoneNumber" attributes are made accessible to all users within the policy tree. The attribute of "userPasswords" is only visible to each respective user to which the value belongs to. Second, each user has write-privileges to themselves for all attributes. Third, users have read-privileges to other users' attributes (excluding "userPasswords").

Sub-goal 3 requires two steps to be completed in-order. First, the "employee" sub-tree is made accessible to "humanresources". Second, "humanresources" possesses write-privileges to all "employee" sub-tree members.

Sub-goal 4 requires two steps to be completed in-order. First, the "engineering" and "accounting" sub-trees are made accessible to respective "manager" users. Second, each "manager" possesses write-privileges to their respective department sub-tree members.

Sub-goal 5 requires two steps to be completed in-order. First, an "admin" user is declared, with the entire policy tree made available to this user. Second, the "admin"

possesses write-privileges to all entities in the entire policy tree.

It should be noted that Figure 3.11 contains a sub-goal ordering of "PLAN: 1-5 in order". In our Approach-2 analysis of each user's policy implementation, We note which of the 7 policy goals failed first. We then note the sub-goal pertaining to the failed policy goal. This sub-goal determines the final error classification for a given participant's implementation.

3.4.3 Error Categories

The four error categories used in our analysis are based on the four types of "cognitive error" that were identified originally in 1988 by D. A. Norman in "The Psychology of Everyday Things" (pp. 4, 8) [33]. These four cognitive failure types are goal failures, plan failures, action failures, and perception failures (pp. 4) [33]. We discuss our four error categories below.

3.4.3.1 Goal Error

Goal-Stage failures describe cases where goals are not achievable, conflicting, or lost altogether (pp. 4) [33]. This category can also describe goals that go un-activated within a system, or activated at the wrong time (pp. 4) [33]. In the context of our OpenLDAP study, Goal-Stage failures describe OpenLDAP goal implementations that are incorrect or forgotten. More specifically, the implementation can be compiled without error, but it will not fulfill the goal requirements.

3.4.3.2 Plan Error

Plan-Stage failures describe cases where some plan within a system is faulty, incorrect, or impossible (pp. 4) [33]. In the context of this OpenLDAP study, Plan-Stage failures describe OpenLDAP goal implementations with incorrect (or impossible) policy syntax.

3.4.3.3 Action Error

Action-Stage failures describe more spontaneous cases where some slip or lapse results in an error (pp. 4) [33]. In the context of this OpenLDAP study, Action-Stage failures describe OpenLDAP goal implementations with common grammatical syntax errors, misspellings, and sub-tree mix-ups.

3.4.3.4 Perception Error

Perception-Stage failures describe errors stemming from misinterpretation (pp. 4) [33]. In the context of this OpenLDAP study, Perception-Stage failures describe OpenLDAP goal implementations with errors that indicate a misunderstanding of the policy goal instructions by the user. This type of error can also include a misunderstanding of OpenLDAP syntax.

3.5 Error Categorization Data Analysis

There were originally 50 participants in the study. The data of 2 participants was not usable (pp. 7) [26]. Therefore, the total data set consists of 48 participants.

For each participant, we started by reviewing the data collected in prior work [25]. This was done to determine which specific goals failed in the participant's implementation. Then, we applied the HTAs outlined in Approach-1 to each of the failed goal implementations. This allowed us to better identify what aspect of the goal implementation was incorrect. Based on this identification, we would then assign one (or more) of the four error categories to the implementation error for each failed goal.

Once we had applied the Approach-1 error classification to a participant's implementation, we then applied the Approach-2 classification. In this approach, we identified which goal failed first in each participant's implementation. We applied the Approach-2 HTA to pinpoint what aspect of this goal implementation lead to failure. Then, based on this identification, we would then assign one (or more) of the four error categories to the implementation error for the single failed goal. This error classification would therefore be

	Perception	Action	Goal	Plan	Total
Goal 1	13	5	6	2	26
Goal 2	11	8	7	2	28
Goal 3	17	13	11	0	41
Goal 4	22	9	15	3	49
Goal 5	19	12	16	4	51
Goal 6	14	7	16	1	38
Goal 7	13	13	12	2	40
Total	109	67	83	14	273
Summary HTA	28	15	20	7	70

Table 3.2: All errors made by study participants based on the Approach-1 (per-goal) and Approach-2 (goal summary) HTA diagrams, from (pp. 9) [26]

applied to the policy implementation as a whole, instead of on a goal-by-goal basis.

The data in Table 3.2 shows the total number of error instances for both Approach-1 and Approach-2. The data for Approach-1 is summarized by the rows pertaining to goals 1 through 7. The data for Approach-2 is summarized in the "Summary HTA" row.

The data in Table 3.2 shows a high number of errors in comparison to the number of failed goals that were depicted in Figure 3.2 and Figure 3.3 (pp. 9) [26]. The total possible number of goals to be accomplished is 336 (pp. 9) [26]. The total number of successfully implemented goals is 227 (pp. 9) [26]. Therefore, the study has found 109 instances of failed goals. The calculation of these values is shown in Figure 3.12.

The total amount of failures in Table 3.2 is 273 for Approach-1. This is due to the fact that in many instances of failure, "...a participant made more than one error in their attempt to accomplish a goal" (pp. 9) [26].

The goals with the most total errors are goal 3 (41 errors), goal 4 (49 errors) and goal 5 (51 errors). This is consistent with the data shown in Figure 3.2 (pp. 8) [26].

Across all goals in the Approach-1 categorization, perception errors occurred the most. One explanation for this could be that participants had difficulties "...perceiving the se-

Total Possible Number of Goals to be Accomplished:

48 participants × 7 policy goals = 336 possible goals

Total Number of Successfully Implemented Goals:

4(1 goal)
11(2 goals)
3(3 goals)
1(4 goals)
3(5 goals)
9(6 goals)
7(7 goals)
\+ 10(7 goals without extra permissions)

227 successfully implemented goals

Total Instances of Failed Goals:

336 possible goals - 227 successfully implemented goals = 109 failed goals

Figure 3.12: Calculation of Instances of Failed Goals based on Approach-1 Error Categorization

mantics or consequence of a particular directive in an OpenLDAP ACL..." (pp. 9) [26]. Participants may have misunderstood the written policy goal, how to implement such a goal within the confines of OpenLDAP syntax, or both. Participants may also have struggled to understand the complexities of the OpenLDAP ACL precedence rule (pp. 9) [26]. For the Approach-2 error categorization, the most common error we observed is also the perception error.

The second-most common error type across all goals in the Approach-1 categorization is the goal error. Within the context of this study, these goal errors could be explained by participants submitting functional implementations that compiled but did not achieve a policy goal. This may have been because the participant forgot to implement a policy goal, or the goal implementation fell short of requirements. In the error categorization process, many failed goals have been assigned multiple categorizations. In many cases, a failed goal is categorized as a goal error and some other additional category (e.g. perception error).

The goal error categorization signifies a failure to meet a goal requirement. The other additional categorization is used to more accurately describe the nature of the observed error.

The third-most common error type across all goals in the Approach-1 categorization is the action error. The prevalence of this type of error can likely be attributed to "...the complex syntax of the OpenLDAP ACLs, even the limited syntax we adopted" (pp. 9) [26]. This error categorization is applied to instances of syntax mistakes, for example.

The least common error is the plan error. The low number of this type of error could be explained by the fact that the study's sample policy does not appear to have any obvious contradictions that would result in impossible policy syntax. During the categorization process, the instances of plan errors largely occurred when there was incorrect policy syntax. The low number of plan errors may indicate a strength of OpenLDAP. While it does have poor usability, it appears that OpenLDAP does not easily allow for policy contradictions.

The prevalence of the perception error over the other three categories points to conceptual misunderstanding as a key reason for the poor usability of OpenLDAP. There is not enough clarity in the tree structure of OpenLDAP ACLs, and how to implement policies that take on such a structure. This disconnect between a written policy and its implementation in OpenLDAP ACLs means that a user's idea of how OpenLDAP works may not match how it actually works. Consequently, the user's policy intentions can get lost in translation during the implementation. A redesign of OpenLDAP would be necessary in order to reduce this gap and improve usability.

Our findings from the human participant study show that OpenLDAP software has poor usability. The error categorizations (via Approach-1 and Approach-2) discussed in this section have been useful in determining specific causes of this poor usability.

Chapter 4

Conclusions

In this chapter, we summarize the conclusions for two projects: (i) an implementation of the 5G-AKA protocol, and, (ii) a usability analysis of OpenLDAP ACLs using HTA. We discuss the conclusions for (i) in Section 4.1. We discuss the conclusions for (ii) in Section 4.2.

4.1 5G-AKA Implementation

Our implementation of the 5G-AKA protocol using the Java programming language serves as a proof-of-concept for the authentication method. There is an initiation phase where the UE subscriber sends encrypted identifier information to the SN, eventually reaching the HN. This is followed by the 12-step process of the 5G-AKA protocol. Our implementation has demonstrated that the 5G-AKA protocol as specified by the 3GPP does indeed work as per its cryptographic specifications. This was verified through manual testing to confirm that the SUPI value was successfully decrypted at the end of the initiation phase. This was also verified through manual testing to confirm that the coincidence checks in steps 9 and 11 were successful.

Our implementation performs the protocol in a clear and understandable way. Each step of the protocol is displayed to a command-line terminal. This includes a description of each step, and the outputs of each cryptographic function.

Our implementation also includes error handling functionality in accordance with 3GPP specifications. If a MAC failure occurs, then the relevant message is sent from the UE to the SN. If a synchronization failure occurs, then a multi-step process to reset SQN_{HN} is performed.

The 3GPP 5G-AKA specification has been found to have a shortcoming in terms of its security. The UE does not get any notification of successful or unsuccessful authentication from the SN or HN. This flaw was originally found by Basin et al., described in [24]. Basin et al. proposed two possible solutions to this issue [24]. We have implemented both solutions as part of our 5G-AKA protocol implementation. Each of these UE-side confirmation solutions can be enabled within our code using Boolean flags.

The long-term goal of the 5G-AKA implementation is to release it as an open-source project on a public repository. This will allow researchers to inspect and tinker with the source code. People will be able to point out potential areas of weakness in the implementation, and propose new ideas to strengthen the authentication protocol. We argue that the MIT license would be the best suited license for this open source project.

In conclusion, our 5G-AKA implementation satisfies the requirements described earlier in Section 2.2. Our implementation authenticates a UE based on the 3GPP protocol specifications, and it communicates each step to a command-line terminal console. We discuss a summary of future work for our 5G-AKA implementation in Chapter 5.

4.2 Usability Analysis of OpenLDAP ACLs

OpenLDAP is a protocol that allows system administrators to explicitly assign resource permissions to specific users within a directory system.

Based on prior work discussed in [25] and [26], it was determined that OpenLDAP software has poor usability. A human-subject study was carried out to observe participants' attempts to implement a system policy using OpenLDAP ACLs.

In order to better understand the causes for the poor usability of OpenLDAP, error categorization is necessary. The approach of creating HTA diagrams based upon THEA

concepts has proved useful in this endeavor. We have created two sets of HTA diagrams for error categorization. We identify them as Approach-1 and Approach-2, respectively.

Approach-1 uses seven separate HTA diagrams, one for each of the seven policy items in the sample policy goals that participants were required to implement in the human-subject study. These HTA diagrams break down the goal of each policy item into a set of sub-goals to be completed in some specified order. We have applied the Approach-1 HTAs to each human-subject study participant's policy implementation in order to better pinpoint sources of errors in their respective failed goals.

Approach-2 uses a single HTA diagram. This diagram groups the policy items into sub-goals. This diagram's main goal is the overall correct implementation of all the seven policy items in the study's sample policy. In Approach-2, we determined which of the seven policy items failed first in a submitted implementation. That goal was considered the singular error of the failed OpenLDAP policy. Error categorization was applied to that single failed goal.

Based upon THEA concepts, four types of errors were used in the categorization. They are as follows: goal errors, plan errors, action errors, and perception errors.

Upon completion of error categorization, it was found that in both Approach-1 and Approach-2, perception errors were the most common. Considering the Approach-1 data in particular, we proposed that the complexity of OpenLDAP syntax and its precedence rule may be the cause for the higher incidence of perception errors. We recommended that OpenLDAP should undergo some form of redesign in order to improve usability and decrease the prevalence of errors such as perception errors. We discuss this further in Chapter 5.

Chapter 5

Future Work and Improvements

In this chapter, we summarize the future work and potential improvements for two projects: (i) an implementation of the 5G-AKA protocol, and, (ii) a usability analysis of OpenLDAP ACLs using HTA. We discuss future open-source work and protocol improvements for (i) in Section 5.1. We discuss potential redesign options for (ii) in Section 5.2. The purpose of these redesigns for (ii) would be to improve OpenLDAP usability.

5.1 5G-AKA Implementation

As we have mentioned previously in this thesis, the 5G-AKA protocol implementation will eventually be released as an open-source project on a public code repository. This will allow for other developers and researchers to tinker with and improve the 5G-AKA implementation. A Bring-Your-Own-SUPI (BYOSUPI) feature will be included as part of the open-source 5G-AKA protocol implementation. This is a proposed improvement upon the standard 5G-AKA protocol.

The primary goal of our implementation was for it to be a proof-of-concept of the 5G-AKA protocol. Hence, our main focus was the development of a working authentication process. Future work will include improving the formatting of our implementation code. We will improve our code using a standard naming convention for methods and variables. This will improve the code's organization and clarity.

5.2 Usability Analysis of OpenLDAP ACLs

Redesigning OpenLDAP software to improve usability would be a very large and complicated undertaking. In such a design, the tree-structure approach to storage in LDAP must be "...carefully accounted for in any redesign..." (pp. 9) [26]. Another aspect of a redesign would be to tackle the issue of complex syntax rules in OpenLDAP ACLs (pp. 9) [26].

Simplifying the OpenLDAP syntax could reduce the instances of action errors (pp. 9) [26]. Syntax simplification could be especially useful in reducing instances of grammatical syntax errors in OpenLDAP policies. It could also help reduce perception errors, since an easy-to-use syntax would reduce incorrect perceptions of how the OpenLDAP protocol is structured, how it functions, and so on.

An observation was made that errors were made by participants when multiple policy goals sought to "...address the same resources..." (pp. 9) [26]. One possible redesign would be to ensure that for "...any resource, there is allowed to exist at most one directive in the ACL that pertains to it" (pp. 9) [26]. Such a redesign could simplify the precedence rule of OpenLDAP, in turn potentially reducing the rate of perception errors (pp. 9) [26].

References

[1] "ETSI TS 133 501; 5G; Security architecture and procedures for 5G System (3GPP TS 33.501 version 16.3.0 Release 16)," ETSI 3GPP, Report V16.3.0, 2020, pp. 21-25, 31, 32, 37, 39, 40, 43-45, 73, 98, 181, 183, 191-193, 210-212.

[2] "3GPP TS 24.501; 3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Non-Access-Stratum (NAS) protocol for 5G System (5GS); Stage 3; (Release 16)," ETSI 3GPP, Report V16.5.1, 2020, pp. 464, 521, 523, 524, 577.

[3] "3GPP TS 23.003; 3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Numbering, addressing and identification; (Release 16)," ETSI 3GPP, Report V16.2.0, 2020, pp. 19-21, 50, 52.

[4] "3GPP TS 33.501; 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Security architecture and procedures for 5G system (Release 17)," ETSI 3GPP, Report V17.1.0, 2021, pp. 211-215.

[5] "ECDSA signature generation using secp256r1 curve and SHA256 algorithm - BouncyCastle,"

url: https://stackoverflow.com/questions/25261823/ecdsa-signature-generation-using-secp256r1-curve-and-sha256-algorithm-bouncyca, Stack Overflow Question, Answered by users: user3932793, dave_thompson_085 et al.

[6] "Java Security Standard Algorithm Names,"

url: https://docs.oracle.com/en/java/javase/11/docs/specs/security/standard-names.html.

[7] "Java Code Examples for javax.crypto.KeyAgreement - Example 1,"

url: https://www.programcreek.com/java-api-examples/?api=javax.crypto.KeyAgreement.

[8] "After ECDH with Curve25519, is it pointless to use anything stronger than AES-128?" url: https://crypto.stackexchange.com/questions/59762/after-ecdh-with-curve25519-is-it-pointless-to-use-anything-stronger-than-aes-12, Stack Exchange Question, Answered by users: knaccc et al.

[9] B. Buchanan, "You've Heard of X25519, But What's So Special About X448?" April 2020, url: https://medium.com/asecuritysite-when-bob-met-alice/youve-heard-of-x25519-but-what-s-so-special-about-x448-c790ef57ceb1.

[10] *XIP4001C: X25519 Curve25519 Key Exchange IP Core*, XIPHERA, March 2021, Volume 1.0, url: https://xiphera.com/product_brief/XIP4001C_PB.pdf .

[11] "Class ECDHCBasicAgreement," url: https://people.eecs.berkeley.edu/~jonah/bc/org/bouncycastle/crypto/agreement/ECDHCBasicAgreement.html.

[12] "Using elliptic curve Diffie-Hellman with cofactor key for generating symmetric key," url: https://stackoverflow.com/questions/24825177/using-elliptic-curve-diffie-hellman-with-cofactor-key-for-generating-symmetric-k, Stack Overflow Question, Answered by users: Manu et al.

[13] "Convert from big Endian to Hex String with java,"

url: https://stackoverflow.com/questions/40422087/convert-from-big-endian-to-hex-string-with-java, Stack Overflow Question, Answered by users: user567, xro7 et al.

[14] C. Wilson, "Understanding Big and Little Endian Byte Order - Base 16 Hex Numbers," August 2018, url: https://www.digital-detective.net/understanding-big-and-little-endian-byte-order/.

[15] "Java - parse and unsigned hex string into a signed long,"

url: https://stackoverflow.com/questions/5723579/java-parse-and-unsigned-hex-string-into-a-signed-long, Stack Overflow Question, Answered by users: Peter, WhiteFang34 et al.

[16] L. Gupta, "Java Secure Hashing – MD5, SHA256, SHA512, PBKDF2, BCrypt, SCrypt - Medium password security using SHA algorithms - Java SHA Hashing Example," url: https://howtodoinjava.com/security/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples/#sha.

[17] "SHA-256 and SHA3-256 Hashing in Java," url: https://www.baeldung.com/sha-256-hashing-java.

[18] "Online Hex Converter," url: https://www.scadacore.com/tools/programming-calculators/online-hex-converter/.

[19] "Class Cipher," url: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/javax/crypto/Cipher.html.

[20] L. Gupta, "Java AES 256 Encryption Decryption Example,"
url: https://howtodoinjava.com/security/aes-256-encryption-decryption/.

[21] "Cipher Algorithm Modes," url: https://docs.oracle.com/en/java/javase/11/docs/specs/security/standard-names.html#cipher-algorithm-modes.

[22] "Compute HMAC-SHA512 with secret key in java,"
url: https://stackoverflow.com/questions/39355241/compute-hmac-sha512-with-secret-key-in-java, Stack Overflow Question, Answered by users: PowerFlower, dvsakgec, et al.

[23] "3GPP TS 33.102; 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Security architecture (Release 16)," ETSI 3GPP, Report V16.0.0, 2020, pp. 11, 22, 24, 25, 73.

[24] D. Basin, S. Radomirović, J. Dreier, R. Sasse, L. Hirschi, and V. Stettler, "A Formal Analysis of 5G Authentication," in *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada*. New York, NY, USA: ACM, 2018, pp. 1383, 1385, 1387, 1393, 1394.

[25] Y. F. Chen, "Usability of the Access Control System for OpenLDAP," MASc Thesis, University of Waterloo, 2019, published on UWSpace.
url: http://hdl.handle.net/10012/15083. pp. 12, 16, 17, 21, 22, 27, 28, 30-34.

[26] M. Tripunitara, Y. F. Chen, and R. Punchhi, "The Poor Usability of OpenLDAP Access Control Lists (ACLs)," 2021, pp. 1–11.

[27] "About 3GPP," url: https://www.3gpp.org/about-3gpp.

[28] "TCP/IP," Encyclopedia Britannica,
url: https://www.britannica.com/technology/TCP-IP.

[29] S. Sriraam, S. Sajeev, R. Joshi, A. Vithalkar, M. Bansal, and H. Jagadeesh, "Implementation of 5G Authentication and Key Agreement Protocol on Xbee Networks," in *2020 12th International Conference on Communication Systems & Networks (COMSNETS)*. New Delhi, Delhi, India: Department Of Computer Science And Engineering, Department Of Electrical Engineering (IIT Dehli), 2020, pp. 696–698.

[30] R. Budhiraja, "Overview of Indigenous 5G Testbed." Kanpur, Uttar Pradesh, India: Department Of Electrical Engineering (IIT Kanpur), 2021,

url: http://home.iitk.ac.in/~rohitbr/pdf/5gtb_overview.pdf.

[31] J. Sermersheim, "Lightweight Directory Access Protocol (LDAP): The Protocol," June 2006, url: https://tools.ietf.org/html/rfc4511, pp. 1.

[32] "OpenLDAP," OpenLDAP Foundation, url: https://www.openldap.org/.

[33] S. Pocock, M. Harrison, P. Wright, and P. Johnson, "THEA: A Technique for Human Error Assessment Early in Design," University of York, Report, 2001, pp. 1-4, 8.

[34] S. Pocock, B. Fields, M. Harrison, and P. Wright, "THEA – A Reference Guide," University of York, Report, 2001, pp. 10, 15, 23.

[35] J. A. del Peral-Rosado, R. Raulefs, J. A. López-Salcedo, and G. Seco-Granados, "Survey of Cellular Mobile Radio Localization Methods: From 1G to 5G," in *IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 20, NO. 2, SECOND QUARTER 2018*, vol. 20, no. 2. Piscataway, NJ, USA: IEEE, 2018, pp. 1127–1131.

[36] CableLabs, "A Comparative Introduction to 4G and 5G Authentication (Winter 2019)," CableLabs, Report N/A, 2019,

url: https://www.cablelabs.com/insights/a-comparative-introduction-to-4g-and-5g-authentication. pp. 1-4, 8.

[37] ITU-R, "Minimum requirements related to technical performance for IMT-2020 radio interface(s)," ITU-R, Report ITU-R M.2410-0, 2017, pp. 2.

[38] "5G - WHAT ARE THE MAIN USAGE SCENARIOS OF 5G?"

url: https://www.etsi.org/technologies/5G?jjj=1615480552531.

[39] "Java Security Overview: Introduction to Java Security,"

url: https://docs.oracle.com/en/java/javase/11/security/java-security-overview1.html#GUID-2EF91196-D468-4D0F-8FDC-DA2BEA165D10.

[40] "Package java.security,"

url: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/security/package-summary.html.

[41] "Package javax.crypto,"

url: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/javax/crypto/package-summary.html.

[42] "N1/N2 Interface," url: https://www.developingsolutions.com/products/dstest-5g-core-network-testing/n1-n2-interface-application/.

[43] "3GPP TS 24.007; 3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Mobile radio interface signalling layer 3; General aspects; (Release 17)," ETSI 3GPP, Report V17.0.0, 2020, pp. 99, 105, 108.

[44] D. R. L. Brown, "Standards for Efficient Cryptography - SEC 1: Elliptic Curve Cryptography," Certicom Research, Report V2.0, 2009, pp. 31-33, 67.

[45] D. J. Bernstein, "Curve25519: new Diffie-Hellman speed records," International Association for Cryptologic Research, Report N/A, 2006, pp. 1.

[46] D. R. L. Brown, "SEC 2: Recommended Elliptic Curve Domain Parameters," Certicom Research, Report V2.0, 2009, pp. 9.

[47] "3GPP TS 33.220; 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Generic Authentication Architecture (GAA); Generic Bootstrapping Architecture (GBA) (Release 17)," ETSI 3GPP, Report V17.0.0, 2020, pp. 40.

[48] "LTE Quick Reference - Authentication - Example," ShareTechNote,

url: https://www.sharetechnote.com/html/Handbook_LTE_Authentication.html#Key_Generation_TestUSIM_Example1.

[49] "APACHE LICENSE, VERSION 2.0," The Apache Software Foundation,

url: https://www.apache.org/licenses/LICENSE-2.0.

[50] "GNU General Public License," Free Software Foundation,

url: https://www.gnu.org/licenses/gpl-3.0.en.html.

[51] "The MIT License," Open Source Initiative Website,
url: https://opensource.org/licenses/MIT.

[52] J. H. Saltzer, "The Origin of the "MIT License"," *IEEE Annals of the History of Computing*, pp. 94–98, November 2020, doi: 10.1109/MAHC.2020.3020234.

[53] *OpenLDAP Software 2.5 Administrator's Guide*, OpenLDAP Foundation, February 2021, Volume 2.5, url: https://www.openldap.org/doc/admin25/OpenLDAP-Admin-Guide.pdf. pp. 3.

[54] I. Bente, "Keeping your sanity while designing OpenLDAP ACLs," November 2016,
url: https://medium.com/@moep/keeping-your-sanity-while-designing-openldap-acls-9132068ed55c.