Politecnico di Bari

Department of Electrical and Information Engineering

Master's Degree in Telecommunications Engineering

Dissertation in

WIRELESS NETWORK SECURITY

A LOOKING-FORWARD LAWFUL INTERCEPTION ARCHITECTURE FOR FUTURE MOBILE COMMUNICATION SYSTEMS

Supervisor

Prof. Ing. Giuseppe Piro, PhD

Candidate

Marco Olivieri

Advisor

Ingrid Huso, PhD Student

Academic Year 2023 - 2024

LIBERATORIA ALLA CONSULTAZIONE DELLA TESI DI LAUREA DI CUI ALL'ART.4 DEL REGOLAMENTO DI ATENEO PER LA CONSULTAZIONE DELLE TESI DI LAUREA (D.R. n. 479 del 14/11/2016).

Il sottoscritto **OLIVIERI MARCO** matricola **589630**

Corso di Laurea **INGEGNERIA DELLE TELECOMUNICAZIONI**

autore della presente tesi di Laurea dal titolo:

A LOOKING-FORWARD LAWFUL INTERCEPTION ARCHITECTURE FOR FUTURE MOBILE COMMUNICATION SYSTEMS

Parola chiave: Lawful Interception, national security, Beyond 5G, key escrow, Edge Computing

Abstract:

The 5th Generation (5G) of mobile networks is the latest evolution of telecommunications networks designed to guarantee faster and more reliable connections. 5G technology introduces new and more stringent standards of security and privacy for users, including the use of end-to-end encrypted connections. However, this technology has raised critical issues in Lawful Interception (LI), posing a serious obstacle to intelligence activities for national security, particularly in detecting criminal activities. The rise of cybercrime within the European Union (EU) has highlighted the problem of implementing a Lawful Interception architecture that allows the Law Enforcement Agencies (LEAs) to retrieve and decrypt end-to-end encrypted traffic of users under investigation. In response, this thesis aims to propose and validate a real-time Lawful Interception framework that intercepts traffic between two users connected to the same 5G network, by implementing key escrow mechanisms. Two implementation methodologies of the proposed LI framework are considered: the first uses LI operations in the Core Network (CN), while the second aims to analyse a cutting-edge solution which exploits the benefits of shifting the LI workload to the edge of the network.

☒ Autorizza

□ Non autorizza

la consultazione della presente tesi, fatto divieto a chiunque di riprodurre in tutto o in parte quanto in essa contenuto.

Bari, 25/07/2024

Firma Marco Olivieri

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

Acronym	Definition	Acronym	Definition
2G	2^{nd} Generation	**IMSI**	International Mobile Subscriber Identity
3G	3^{rd} Generation	**IoT**	Internet of Things
3GPP	3^{rd} Generation Partnership Project	**IP**	Internet Protocol
4G	4^{th} Generation	**IQF**	Identifier Query Function
5G	5^{th} Generation	**IRI**	Intercept Related Information
5G NR	5G New Radio	**KPI**	Key Performance Indicator
5GC	5G Core Network	**LEA**	Law Enforcement Agency
AAA	Authentication, Authorization, and Accounting	**LEMF**	Law Enforcement Monitoring Facility
ADMF	Administration Function	**LI**	Lawful Interception
AES	Advanced Encryption Standard	**LICF**	Lawful Interception Control Function
AF	Application Function	**LIID**	Lawful Interception Identifier
AI	Artificial Intelligence	**LIPF**	Lawful Interception Provisioning Function
AKA	Authentication and Key Agreement	**LTE**	Long-Term Evolution
AMF	Access and Mobility Management Function	**MAC**	Medium Access Control
AOR	Address of Record	**MCC**	Mobile Country Code
API	Application Programming Interface	**MDF**	Mediation and Delivery Function
ARPF	Authentication Repository and Policy Function	**ME**	Mobile Equipment
ARQ	Automatic Repeat reQuest	**MEC**	Mobile Edge Computing
AS	Access Stratum	**MIMO**	Multiple-Input Multiple-Output
AUSF	Authentication Server Function	**mIoT**	Massive Internet of Things
BSF	Binding Support Function	**MITM**	Man-in-the-Middle
BWP	Bandwidth Part	**mMTC**	Massive Machine Type Communication
CA	Certificate Authority	**mmWave**	Millimeter Wave
CC	Communication Content	**MNC**	Mobile Network Code
CDMA	Code Division Multiple Access	**MO**	Mobile Operator
CN	Core Network	**MSIN**	Mobile Subscriber Identification Number
CP	Control Plane	**N3IWF**	Non-3GPP Interworking Function
CSP	Communication Service Provider	**NAS**	Non-Access Stratum
CUPS	Control/User Plane Separation	**NAT**	Network Address Translation
DN	Data Network	**NEF**	Network Exposure Function
DRA	Diameter Routing Agent	**NF**	Network Function
E2EE	End-to-End Encryption	**NFR**	Network Function Repository
EAP	Extensible Authentication Protocol	**NFV**	Network Functions Virtualization
eMBB	Enhanced Mobile Broadband	**NGAP**	Next Generation Application Protocol
EPS	Evolved Packet System	**NR**	New Radio
ETSI	European Telecommunications Standards Institute	**NSA**	Non-Standalone
EU	European Union	**NSSAI**	Network Slice Selection Assistance Information
FEC	Forward Error Correction	**NSSF**	Network Slice Selection Function
gNB	Next Generation Node B	**OFDM**	Orthogonal Frequency Division Multiplexing
GTP	GPRS Tunneling Protocol	**OFDMA**	Orthogonal Frequency Division Multiple Access
GUI	Graphical User Interface	**OTR**	Off-the-Record
HARQ	Hybrid Automatic Repeat reQuest	**PCF**	Policy Control Function
HI	Header Interface	**PDCP**	Packet Data Convergence Protocol
IDBC	ID-based cryptosystem	**PDU**	Protocol Data Unit
IM	Instant Messaging	**PKI**	Public Key Infrastructure
IMEI	International Mobile Equipment Identity	**POI**	Point of Interception

Acronym	Definition	Acronym	Definition
QoS	Quality of Service	**SMSF**	Short Message Service Function
RADIUS	Remote Authentication Dial-In User Service	**SRTP**	Secure Real-time Transport Protocol
RAN	Radio Access Network	**SUCI**	Subscriber Concealed Identifier
RAT	Radio Access Technology	**TCP**	Transmission Control Protocol
REST	REpresentational State Transfer	**TDD**	Time Division Duplexing
RLC	Radio Link Control	**TF**	Triggering Function
RRC	Radio Resource Control	**TKA**	Trusted Key Authority
RTCP	Real-time Transport Control Protocol	**TLS**	Transport Layer Security
RTP	Real-Time Protocol	**TTI**	Transmission Time Interval
SA	Standalone	**TTP**	Trusted Third-Party
SBA	Service-Based Architecture	**UDM**	Unified Data Management
SC-FDMA	Single Carrier Frequency Division Multiple Access	**UDP**	User Datagram Protocol
SCS	Subcarrier Spacing	**UE**	User Equipment
SCTP	Stream Control Transmission Protocol	**UI**	User Interface
SDAP	Service Data Application Protocol	**UMTS**	Universal Mobile Telecommunications System
SDES	Session Description Protocol Security Descriptions	**UP**	User Plane
SDN	Software Defined Networking	**UPF**	User Plane Function
SDU	Service Data Unit	**URLLC**	Ultra-Reliable Low Latency Communication
SEAF	Security Edge Access Function	**USIM**	Universal Subscriber Identity Module
SIM	Subscriber Identity Module	**VNF**	Virtualized Network Functions
SIP	Session Initiation Protocol	**VoIP**	Voice over IP
SIRF	System Information Retrieval Function	**xCC**	Communication Content
SMF	Session Management Function	**xIRI**	Intercept Related Information
SMS	Short Message Service		

INTRODUCTION

The 5^{th} Generation (5G) of mobile networks, standardized by the 3^{rd} Generation Partnership Project (3GPP), is currently the latest evolution of telecommunications networks, designed to guarantee faster and more reliable connections than the previous generation, and it is applied in the most futuristic scenarios such as autonomous driving or remote surgery. In addition to high communication capacity (i.e., enhanced data rate and reduced latency), 5G technology introduces new and more stringent standards of security and privacy for users, including the usage of end-to-end encrypted connections. Although the usage of End-to-End Encryption (E2EE), for instance in well-known messaging apps (e.g., WhatsApp, Telegram, Signal, Skype, etc.) or Voice over IP (VoIP) calling services, is perceived as an undisputed advantage by the majority of users, in reality this aspect has created critical issues in the area of LI. Specifically, the same technology that guarantees and enhanced level of confidentiality between users is posing a serious obstacle to intelligence activities for national security, particularly with regard to the detection of criminal activities, such as the organization of an attack or the sale of smuggled goods. The rise of cybercrime within the European Union (EU) has highlighted the problem of implementing a Lawful Interception architecture that, under a warrant and in compliance with privacy directives, makes the Law Enforcement Agency (LEA) able to retrieve and decrypt the end-to-end encrypted traffic of the users under investigation.

According to the highlighted critical issues, the purpose of this thesis project is to propose and then validate an innovative and 3GPP-compliant Lawful Interception framework that allows intercepting the traffic between

two users connected to the same 5G network, by implementing key escrow mechanisms to decrypt end-to-end encrypted streams. In the study, two implementation methodologies of the proposed LI framework are considered: in the first one, LI operations are performed in the Core Network (CN), while a second cutting-edge proposal aims at analyzing the benefits of shifting the LI workload to the Radio Access Network (RAN).

Firstly, Chapter 1 presents the standard underlying LI and the 5G architecture and protocol stack. Furthermore, it describes the integration of LI in 5G environments, investigating the challenges posed by end-to-end encryption and introducing Key Escrow.

Subsequently, Chapter 2 describes in detail the features of the proposed LI framework, starting with the presentation of the used software and ending with the description of the implementation procedure used for the 5G Core Network (5GC), 5G RAN and the LI architecture both in the Core and in the RAN. It also shows the steps for configuring the system that emulates the interception of a data flow exchanged between two users during a VoIP call or file sharing (e.g., media files or text messages).

Finally, Chapter 3 aims to evaluate the performance of the two implementations of the developed LI framework, examining the latency time of the interception procedure of a file or a VoIP call, as well as providing an assessment of the impact of the LI framework on the Quality of Service (QoS) perceived by users during the communication. To conclude, an objective analysis of the obtained results is carried out in order to show how the proposed LI framework is an efficient response to the critical security issues posed by future mobile telecommunications systems. Moreover, ideas for future developments and applications of the proposed LI framework are provided.

1 LAWFUL INTERCEPTION AND 5G

1.1 LAWFUL INTERCEPTION

Lawful Interception (LI) consists of intercepting and being able to decipher communications between users within a network, sharing them with authorized Law Enforcement Agencies (LEAs). Examples of LEAs include state and federal police, intelligence agencies, or independent anti-corruption commissions. LI is used as a tool in criminal and security investigations, as it can help to gather evidence for legal proceedings, such as identifying networks of relationships among suspected criminals. The ability to provide an adequate LI capability is a requirement for a telecommunications company to obtain a license to publicly offer its services. However, the legitimacy of the interceptions remains a contentious issue: generally, the justification for allowing government interception capabilities is based on crime prevention, while objections mostly concern privacy and security threats [1].

1.1.1 LI standardized architecture

The 3GPP defines the Lawful Interception (LI) standard, as part of the mobile network. Specifically, TS 33.126 [1], TS 33.127 [2] and TS 33.128 [3] standardize the requirements, the architecture and the procedures, respectively. The high-level architecture of LI can be observed in Figure 1.1. The functional entities within this architecture include [2]:

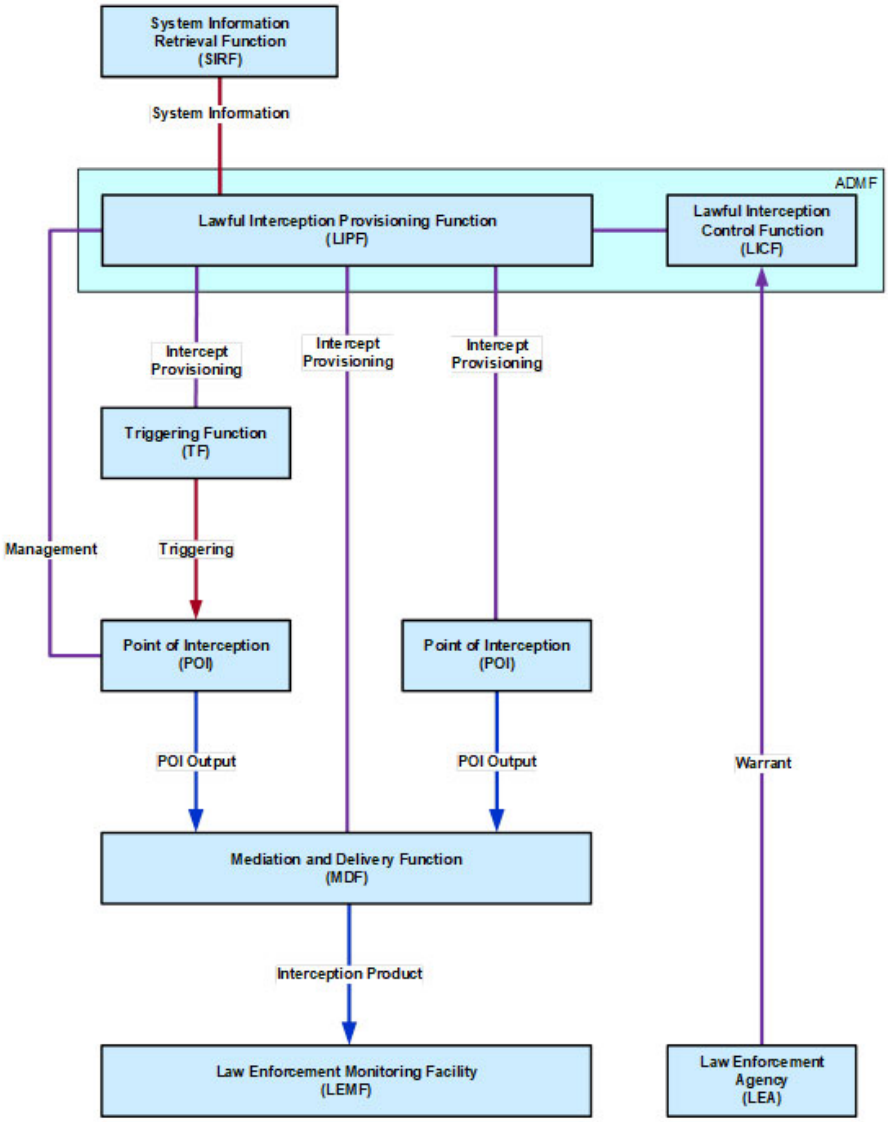


Figure 1.1: Generic view of the LI architecture [1].

- **Point of Interception (POI)**: it detects target communications and extracts information from them. In particular, the intercepted information comprises two types: Intercept Related Information (IRI), which encompasses metadata of the communication (such as users, time, duration, data type, etc.), and Communication Content (CC), which refers to the actual data exchanged during interception. POIs do not directly deliver IRI messages or CC; instead, they transmit Intercept Related Information (xIRI) and Communication Content (xCC),

which need to be further processed before acquiring legal validity, thus becoming ready to be transmitted to the LEA. The outcome of a POI is controlled by the type of Network Function (NF) associated with it, and they can be either embedded within a NF or separate from it, depending on their association. POIs are categorized into two types:

  - Directly provisioned POIs, deployed by the Lawful Interception Provisioning Function (LIPF), detect target communications requiring interception and extract xIRIs or xCCs based on the POI type.
  - Triggered POIs, activated by a Triggering Function (TF), detect target communications based on triggers.

- **Administration Function (ADMF)**: it manages the administrative and operational aspects of the Communication Service Provider (CSP)'s LI capability. This encompasses overall responsibility for provisioning, modifying, and deactivating POIs, TFs, and Mediation and Delivery Functions (MDFs). ADMF consists of four logical sub-functions:
  - **Lawful Interception Control Function (LICF)**: it handles the end-to-end life cycle of a warrant, providing intercept information derived from the warrant for provisioning at POI, TF, MDF2, and MDF3. Communication between LICF and other entities, except the LEA, is mediated by LIPF.
  - **Lawful Interception Provisioning Function (LIPF)**: it provisions applicable POIs, TFs, and MDFs. Its role varies based on the implementation of network functions and ADMF (e.g., virtual or non-virtual).
  - **Identifier Query Function (IQF)**: it receives and responds to real-time queries from LEAs for identifier associations.

  - **Certificate Authority (CA)**: a trusted entity that archives, signs and releases digital certificates for authentication.

- **Triggering Function (TF)**: provisioned by the LIPF, it is responsible for activating triggered POIs in response to network and service events that match criteria established by the LIPF. Whenever a Triggering Function detects a communication that matches with an active warrant, it sends triggers to the associated triggered POI.

- **Mediation and Delivery Function (MDF)**: it facilitates the delivery of the Interception Product to the Law Enforcement Monitoring Facility (LEMF). This function includes MDF2, responsible for generating IRIs from xIRIs, and MDF3, which generates CCs from xCCs. Subsequently, IRIs and CCs are transmitted to the intercepting LEMF.

- **Law Enforcement Agency (LEA)**: it is in charge of submitting warrants to the CSP, although in some places, warrants might be issued by another legal entity, like the judiciary.

- **System Information Retrieval Function (SIRF)**: it provides LIPF with system-related information about the NFs recognized by the SIRF (such as service topology). This information allows the LIPF/LICF to carry out essential operations in order to set up and support the interception (for instance, provisioning POIs, TFs, and MDFs over the LI_X1 interface).

- **Law Enforcement Monitoring Facility (LEMF)**: it receives the Interception Product (i.e., IRIs and CCs).

Moreover, the 3GPP TS specifications (i.e., [1], [2], and [3]) delineate also the necessary point-to-point interfaces used for the LI function. These interfaces are defined based on their role within the architecture and are illustrated in Figure 1.2. Among them, LI_HI2 and LI_HI3 are of utmost importance: LI_HI2 is utilized to transmit IRIs from MDF2 to LEMF, while LI_HI3 is used to send CCs from MDF3 to LEMF. The packets sent through

these interfaces are provided with an header containing general information like Lawful Interception IDentifier (LIID), timestamp, correlation information, and a payload containing IRIs or CCs.

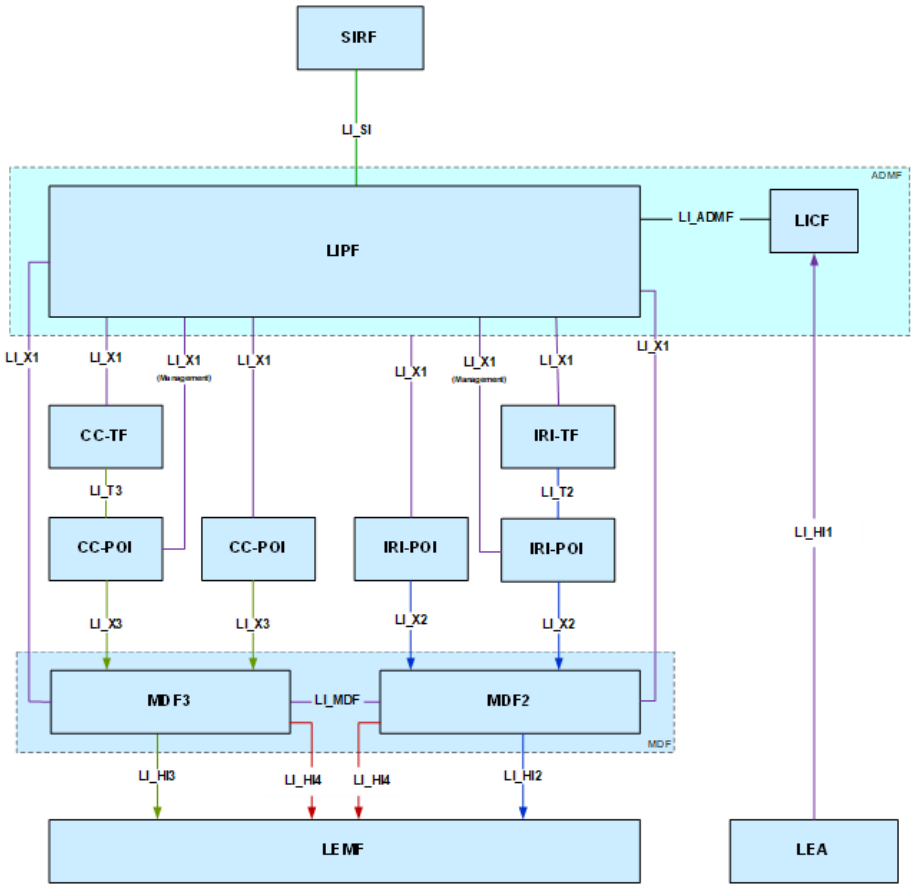


Figure 1.2: High-level interception architecture diagram with key point-to-point LI interfaces [1].

The interception procedure is outlined in 3GPP TS 33.127 [1], and it could be divided into five steps:

1. When targeting a specific User Equipment (UE) for interception, the LEA submits an authorized warrant to the CSP via the LI_HI1 interface.

2. The ADMF receives the warrant via LICF and, by exploiting the LI_ADMF and the LI_X1 interfaces, sends instructions to all participating functions in the LI, setting up TFs and deploying appropriate POIs if needed.

3. Triggered via the LI_T3 interface or directly provisioned by the LIPF, the POIs placed in the User Plane Function (UPF) begin intercepting the communication of the target.

4. The eventually generated xIRI messages and xCC are then transmitted to the corresponding MDFs. Going into detail, the IRI-POI delivers xIRI information through the LI_X2 interface to the MDF2, while the CC-POI delivers xCC data over the LI_X3 interface to the MDF3.

5. Subsequently, the MDF sends IRI messages and CC to the LEMF via the LI_HI2 and LI_HI3 interfaces, respectively.

The LI process is essentially deployed within the domain of Mobile Operators (MOs), empowering them to intercept communications inside their networks. This is the fundamental reason behind the standardized architecture of LI for MOs.

1.1.2 Lawful Interception in 5G Networks

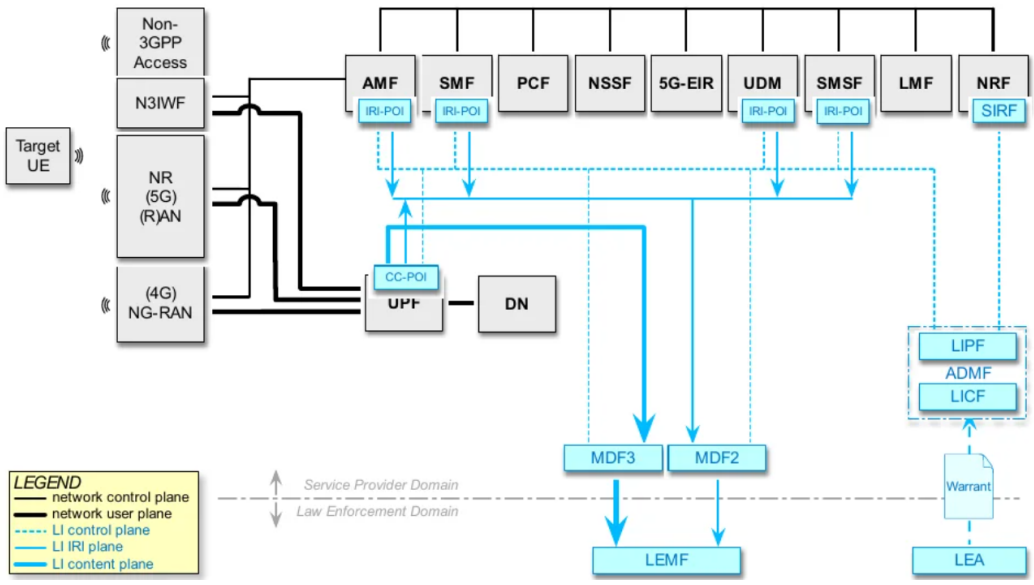


Figure 1.3: Core-anchored LI architecture [1].

In the 5G Core Network architecture, the LI implementation involves the Access and Mobility Management Function (AMF) and the Session Management Function (SMF)/User Plane Function (UPF) [1], as depicted in Figure 1.3.

For the AMF (Figure 1.4a), its LI architecture encompasses capabilities to handle network access, registration, connection management, and the generation of location update information (i.e. xIRIs). Therefore, it acts as a Control Plane Network Function providing xIRI-POI functions. The architecture involves that the LICF in the ADMF receives warrants from LEA and then it provisions the xIRI-POI in the AMF via the LIPF. Moreover, the xIRI-POI in the AMF detects target UE's access and mobility functions and delivers the intercepted information to the MDF, which then forwards it to the LEMF.

On the other hand, regarding the SMF/UPF, as it is illustrated in Figure 1.4b, the LI architecture involves the separation of control plane functions handled by the SMF and user plane data functions handled by the UPF. The SMF includes an xIRI-POI for generating related xIRI, while the UPF includes a xCC-POI for duplicating user plane packets based on interception rules received from the SMF.

Additionally, in the context of LI at the Unified Data Management (UDM) for 5G, the UDM is responsible for managing UE's service area registration-related xIRI.

It is worth mentioning that the standard [1] does not consider the implementation of the LI framework at the edge of the network, e.g. by placing the POI close to the UE in the access network. Therefore, this topic is going to be further explored in Chapters 2 and 3, which are dedicated to the design and the implementation of the cutting-edge proposed LI framework.

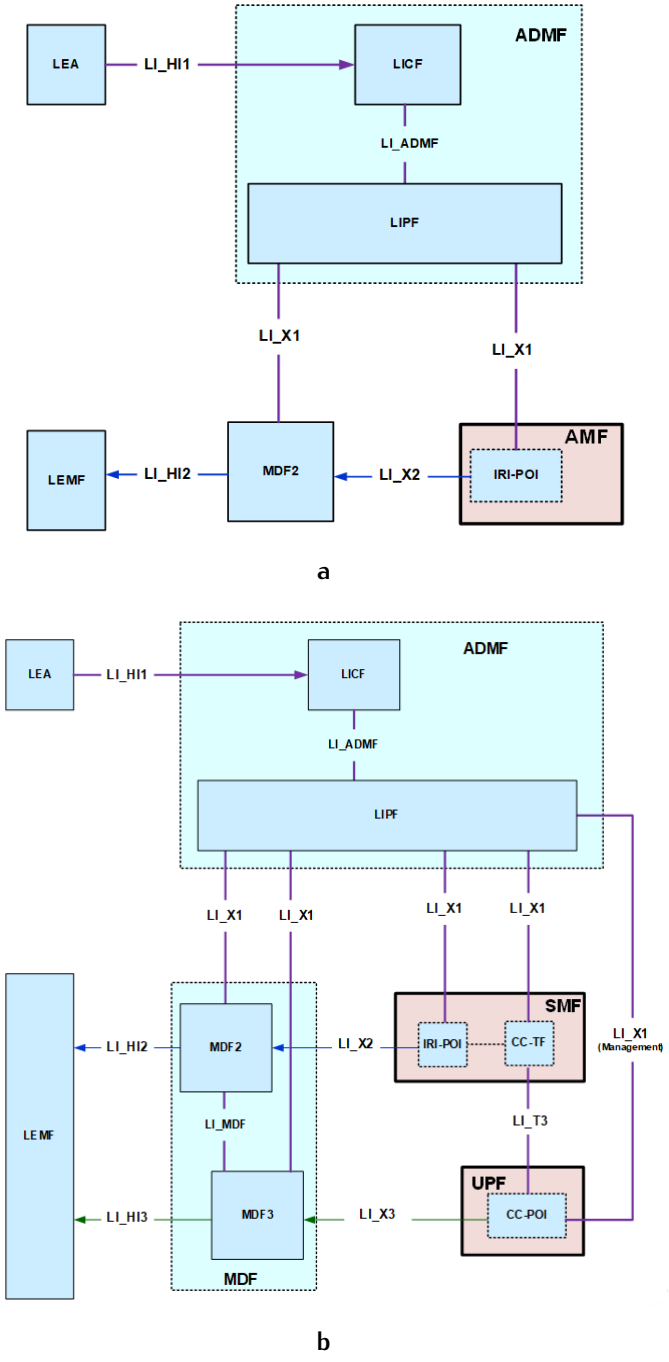


Figure 1.4: Deployment of LI in a 5G network: (a) POIs located in the AMF; (b) POIs located in the SMF and UPF [1].

1.2 5G: FIFTH-GENERATION TECHNOLOGY STANDARD FOR CELLULAR NETWORKS

In the realm of mobile communications, 5G represents the fifth generation of mobile networks, succeeding its predecessor, 4^{th} Generation (4G), and marking a significant evolution in terms of data speed, latency, and device connectivity. Known as New Radio (NR), 5G RAN represents a revolutionary change in mobile network infrastructure, set to redefine interactions with mobile devices and support transformative technologies such as Autonomous Driving, Smart Cities, Industry 4.0, and Virtual Reality. European Telecommunications Standards Institute (ETSI) [4] and 3GPP [5] delineate the primary usage scenarios for 5G:

- **Enhanced Mobile Broadband (eMBB):** it represents an improvement over 4G's mobile broadband, offering higher peak data rates (up to 20 Gbit/s) and user-experienced data rates (up to 100 Mbit/s). It optimizes spectrum usage through innovative coding methods, boosts traffic capacity, and lowers energy consumption. The goal of eMBB is to provide faster internet connectivity for quick data sharing and to handle multiple device interactions simultaneously.

- **Ultra-Reliable Low Latency Communication (URLLC):** tailored to stringent latency and packet loss requirements, URLLC finds application in domains such as autonomous vehicles, mission-critical systems, and industrial automation, all necessitating stable and real-time communication.

- **Massive Machine Type Communication (mMTC):** scalability is crucial to accommodate up to a million devices per square kilometer, as it happens in the Internet of Things (IoT) world. Massive Internet of Things (mIoT) refers to scenarios where the 5G infrastructure must handle dense traffic from various devices.

Furthermore, 5G brings unique features to the network architecture as well, which include network slicing, network capability exposure, scalability, security, and efficient content delivery. Other notable aspects are also Software Defined Networking (SDN), Virtualized Network Functions (VNF), the separation of network functions and business models, and the division of User Plane (UP) and Control Plane (CP), among others to be explored further.

1.2.1 5G Architecture

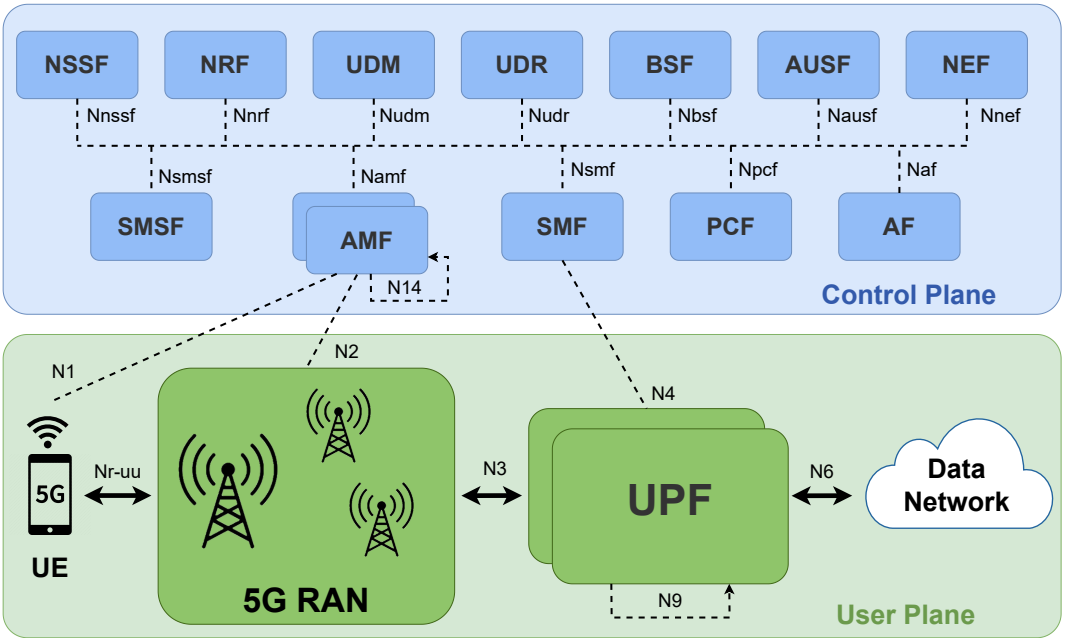


Figure 1.5: 5G architecture and its main network functions.

In the 5G system, entities from the previous generations, such as 4G [6], are kept and improved, including the UE, consisting of a Mobile Station and a Universal Subscriber Identity Module (USIM), along with the RAN. Notably, the architecture of the 5G Core (5GC) is founded upon a Service-Based Architecture (SBA) framework, where the constituent elements are delineated as NFs rather than conventional *Network Entities*. Herein, each NF extends its services to all the authorized NFs and/or to designated consumers allowed to utilize these services. This SBA approach under-

scores modularity and reusability, as depicted in Figure 1.5, which illustrates some functions implemented in the standard [7]:

- **Access and Mobility Management Function (AMF):** it is responsible for ciphering and integrity protection, mobility management, lawful interception, access authentication and authorization, security anchoring, and security context management.
- **Session Management Function (SMF):** it handles session management, allocation and management of UE IP addresses, selection and control of UP functions, policy enforcement, Quality of Service (QoS), and roaming functionality.
- **Authentication Server Function (AUSF):** it manages the authentication and authorization functionalities.
- **Network Exposure Function (NEF):** it provides means to collect, store, and securely expose the services and capabilities provided by NFs (e.g., Application Programming Interfaces (APIs) for NFs).
- **Network Function Repository (NFR):** it maintains and provides information on the deployed NF instances.
- **Policy Control Function (PCF):** it offers a unified policy framework to govern network behaviors, providing policy rules to the control plane functions.
- **Unified Data Management (UDM):** it supports the Authentication Credential Repository and Processing Function, storing long-term security credentials and subscription information.
- **Application Function (AF):** it provides any additional CP function required by specific network slices, potentially supplied by third parties.

- **User Plane Function (UPF):** it acts as an anchor point for inter-Radio Access Technology (RAT) and intra-RAT handovers, packet routing and forwarding. Moreover, the UPF handles User Plane QoS, packet inspection and policy rule enforcement, lawful interception, and traffic accounting and reporting.
- **Network Slice Selection Function (NSSF):** it selects the set of Network Slice instances serving the UE and determining the Allowed Network Slice Selection Assistance Information (NSSAI). In addition, it determines the AMF set to be used to serve the UE, or, based on configuration, a list of candidate AMFs, possibly by querying the NFR.
- **Binding Support Function (BSF):** comparable with the Session Binding Function on the Diameter Routing Agent (DRA) used in 4G, it becomes a mandatory requirement when multiple PCF systems are deployed in the network.
- **Short Message Service Function (SMSF):** the SMSF function is defined in the 5G 3GPP specifications providing Short Message Service (SMS) over Non-Access Stratum (NAS) access to 5G Standalone (SA) devices.

1.2.2 5G Protocol stack

The 5G protocol stack builds upon the established standards of Long-Term Evolution (LTE) while incorporating modifications to better align with the requirements of the new generation in mobile technology. Within 5G, there exist two distinct protocol stacks responsible for managing control messages in the CP and data messages in the UP.

As illustrated in Figures 1.6 and 1.7, the control stacks share common protocols, particularly in the lower layers, while featuring specific layers dedicated to traffic management. Notably, the protocols utilized between User Equipment (UE) and Next Generation Node B (gNB), and between

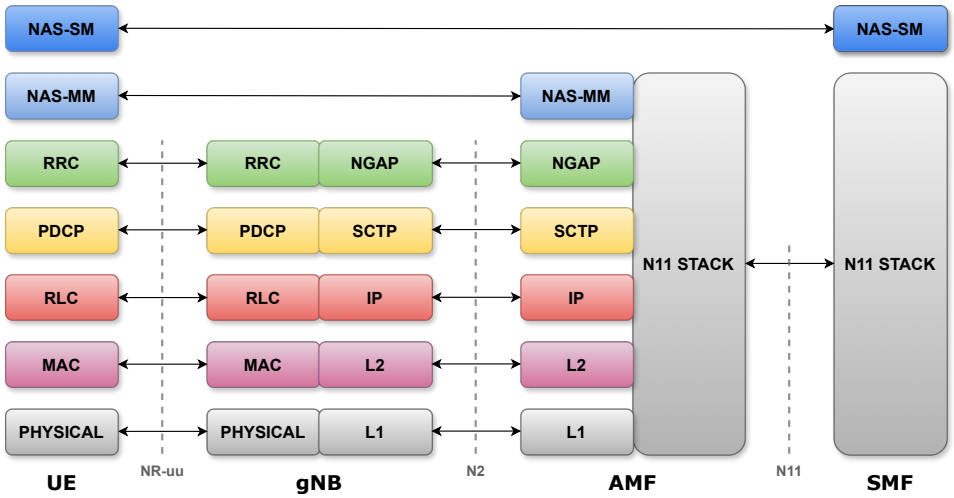


Figure 1.6: 5G Control Plane protocol stack.

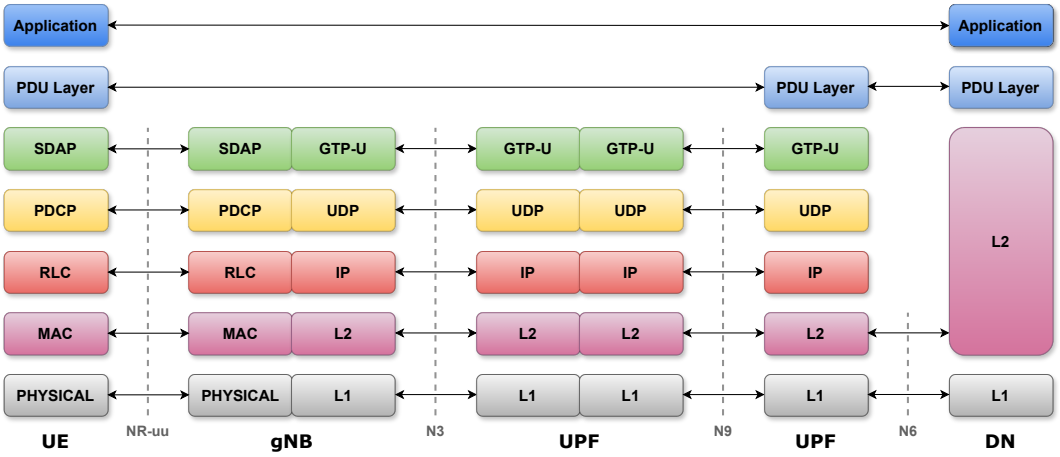


Figure 1.7: 5G User Plane protocol stack.

gNB and the rest of the network, differ. This discrepancy arises from the core network's utilization of internal protocols for data management, with these protocols remaining transparent to end-users due to relaying operations.

Given 5G's integration of network slices to optimize service delivery for various tenants, protocol stacks must adapt accordingly. Consequently, different slices within 5G can possess distinct stack configurations. In some cases, slices may exhibit entirely disparate stacks, rendering each slice a distinct entity. Alternatively, slices may share layers, starting from the lowermost layer. In this scenario, slices may share the same physical layer,

thereby facilitating cooperation in radio resource allocation or sharing common layers up to the PDPC layer, differing only in higher levels.

Regarding individual protocols, the following details, based on [7], are provided, followed by architectural considerations for the stacks:

- **Physical Layer:** it is responsible for radio message transmission between UE and gNB, supporting Orthogonal Frequency Division Multiple Access (OFDMA) and Single Carrier Frequency Division Multiple Access (SC-FDMA) in the uplink, along with hybrid Automatic Repeat reQuest (ARQ), Forward Error Correction (FEC) coding, and modulation.

- **Medium Access Control (MAC) Layer:** it facilitates the mapping of logical connections from upper layers to the physical interface, controlling random access procedures, frame delimitation, station addressing, scheduling information reporting, and message padding.

- **Radio Link Control (RLC) Protocol:** it manages the transport of upper layer Protocol Data Units (PDUs), encapsulated in RLC packets transmitted in Acknowledged, Unacknowledged, and Transparent modes, supporting segmentation and reassembly.

- **Packet Data Convergence Protocol (PDCP):** it controls Service Data Unit (SDU) transmission from upper layers, featuring IP header compression or Robust Header Compression for security via encryption and integrity checks using Advanced Encryption Standard (AES) keys provided by the Authentication and Key Agreement (AKA) procedure. Additionally, PDCP operates in both CP and UP, by using different headers to classify CP and UP data.

- **Radio Resource Control (RRC):** it manages all control signals between UE and gNB exclusively in the CP stack, facilitating radio session establishment and management, radio resource allocation, and the handover procedure.

- **Non-Access Stratum (NAS) Protocol:** it is the highest protocol in the CP stack responsible for UE session and mobility management, connecting the UE to the core network, performing authentication procedures, user registration, IP assignment, and session maintenance. NAS includes NAS-MM and NAS-SM, which communicate with the AMF and SMF, respectively. NAS-MM is also capable of transmitting data not terminating in the AMF.

- **Service Data Application Protocol (SDAP):** it is exclusively present in the UP stack in 5G, and it maps QoS within PDU sessions to specific radio bearers, ensuring that user-specific content is matched to standardized QoS.

- **Stream Control Transmission Protocol (SCTP):** it is the transport protocol for transmitting control messages in the CP of the Core Network, combining the message-oriented feature of User Datagram Protocol (UDP) with in-sequence transport of messages like Transmission Control Protocol (TCP). SCTP supports both ordered and unordered messages and is able to transmit data via multiple paths to add redundancy.

- **Internet Protocol (IP):** it is a ubiquitous network layer protocol with IPv4 and IPv6 versions, responsible for routing data in the UP and CP of the core network.

- **Next Generation Application Protocol (NGAP):** exclusively used in the CP stack, it is the signalling protocol between gNB and AMF supporting configuration updates, UE context transfer, PDU session management, and mobility procedures.

- **User Datagram Protocol (UDP):** it is used in the UP stack to transmit encapsulated messages quickly, through standardized ports 2123 and 2152.

- **GPRS Tunneling Protocol (GTP):** it is a set of IP-based tunneling protocols transporting packets in the core network, including GTP-C (Control) and GTP-U (User data), with GTP-C managing signaling between GTP nodes on port 2123 and GTP-U encapsulating user data in GTP packets sent via UDP on port 2152.

1.2.3 5G key enablers and novelties

The evolution of 5G networks aims to establish a dynamic and flexible network environment capable of supporting different services with their specific performance needs. For this objective to be achieved, a fundamental transformation in network technologies and architecture is required, beginning with the concept of Control/User Plane Separation (CUPS). On this direction, the main novelties introduced with 5G networks are [8]:

- **Network slicing**: this new features in 5G networks divides the physical infrastructure into independent virtual networks, known as network slices, fulfilling different service needs like automotive, mobile broadband, and haptic Internet. These slices enable optimized QoS by accommodating varied demands through independent orchestration.
- **Software Defined Networks (SDN) and Network Functions Virtualization (NFV)**: SDN and NFV are driving 5G evolution, addressing scalability, flexibility, and programmability. NFV leverages cloud computing to Virtualized Network Functions (VNFs), offering unprecedented flexibility, while SDN enables dynamic traffic management for enhanced performance.

In the domain of RAN, 5G brings innovations in radio signal transmission technologies. By following the standard requirements issued by 3GPP, 5G aims to achieve higher speeds, lower power consumption, and increased device density. Notable accomplishments in this field include [9]:

- **Flexible Numerology**: the scalable Orthogonal Frequency Division Multiplexing (OFDM) numerology in 5G New Radio (5G NR) specifies parameters like Subcarrier Spacing (SCS), Transmission Time Interval (TTI), and Cyclic Prefix (CP), with higher numerology indexes corresponding to larger SCSs. Wider SCSs mitigate inter-carrier interference and phase noise at Millimeter Wave (mmWave) frequencies, while also reducing slot duration for lower latencies. As SCS widens, the TTI decreases, impacting system performance parameters and overhead.

- **Bandwidth Part (BWP)**: in 5G NR, BWP defines a fixed band with consistent numerology for transmissions. Each BWP corresponds to a subset of contiguous resource blocks on a carrier, allowing for efficient bandwidth management.

- **Massive Multiple-Input Multiple-Output (MIMO)**: large antenna arrays can be used to transmit many concurrent streams, enabling 3D beamforming.

- **Millimeter wave (mmWave)**: mmWave in the 30-300 GHz range provide wide bandwidth for high data rates in 5G networks. However, they face greater propagation losses outdoor, and encounter blockage indoor. Despite these challenges, mmWaves are advantageous for femtocells and benefit from compact antennas, aiding Massive MIMO deployment.

- **Dynamic Time Division Duplexing (TDD)**: in TDD, the direction of transmission (uplink or downlink) can be flexibly determined, with granularity down to a Mini-slot (2 OFDM symbols). Indeed, the scheduler has the flexibility not only to balance between uplink and downlink transmissions, but also to adjust resource allocation duration.

- **Hybrid Automatic Repeat reQuest (HARQ)**: ARQ improves communication performance by retransmitting erroneously received data. The receiver communicates with ACK/NACK messages to prompt the transmitter's retransmission process. HARQ combines ARQ with FEC techniques, preserving unsuccessful decoding attempts for future joint decoding, and enabling in 5G partial retransmission of the corrupted data.

Moreover, 5G is designed to facilitate the deployment of **Mobile Edge Computing (MEC)** architectures, relocating computational resources to the network periphery to enhance speed, reduce latency, and alleviate core network congestion. By executing operations at the network edge, MEC minimizes data traversal distances, thereby optimizing latency and enhancing user experience. Applications of MEC span diverse domains, including streaming services, where edge servers host frequently accessed content, thereby enhancing accessibility and minimizing reliance on central servers [4].

1.2.4 5G Security

According to the study [11], in the transition from 2^{nd} Generation (2G) to 5G, security remains a crucial concern in mobile systems. The key to build secure systems is ensuring mutual authentication and trust between users and the network, both of which are facilitated by Authentication, Authorization, and Accounting (AAA) mechanisms, providing secure network services and billing for subscribers.

Initially, in 2G systems, user authentication relied on the Subscriber Identity Module (SIM), a secure element housing the subscriber's International Mobile Subscriber Identity (IMSI) and a permanent key. However, the absence of mutual authentication led to vulnerabilities, such as active attacks where an attacker could pose as a valid base station to the subscriber. With the advent of 3^{rd} Generation (3G), the 3GPP introduced Authentication and

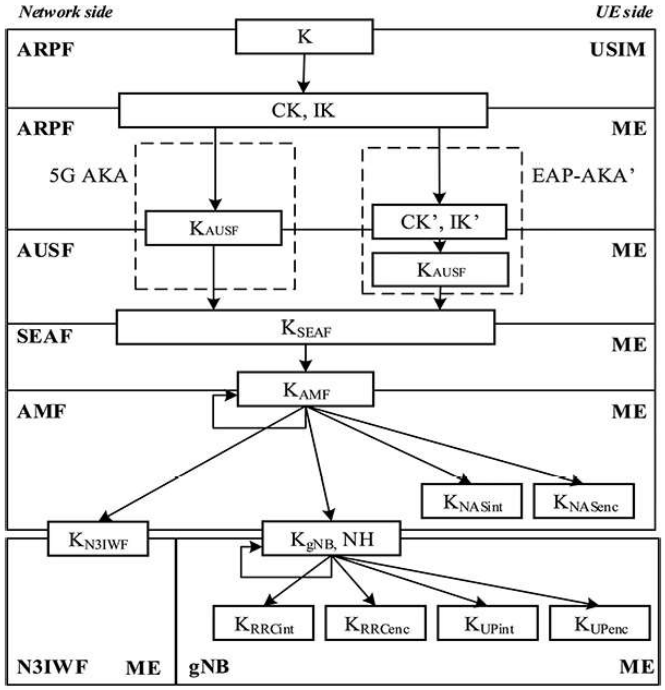


Figure 1.8: 5G key derivation hierarchy [10].

Key-agreement (AKA) protocols with mutual authentication features to mitigate these risks. Notably, the AKA mechanism evolved in 4G systems (Evolved Packet System (EPS)-AKA), with differences compared to its 3G counterpart (Universal Mobile Telecommunications System (UMTS)-AKA) [11].

Looking ahead to the new 5G generation, there is a demand for higher throughput, low latency, and improved quality of service. Thus, new requirements, including IoT connectivity, network slicing for specific customers or vertical sectors, and managing heterogeneous network accesses, are emerging [11]. Following this direction, 3GPP gives technical details about security architecture and procedures for 5G system in TS 33.501 [10]. Therefore, the 5G architecture brings new AAA design choices while preserving some elements from previous generations. There is, indeed, a key hierarchy represented in the Figure 1.8. As described in [12], similar to its function in a 4G system, the long-term secret key (K) provisioned in the USIM and the 5G core network acts as the main source of security context. Unlike LTE, 5G incorporates two types of authentication: primary authen-

tication, mandatory for all devices to access mobile network services, and secondary authentication to an external Data Network (DN), if requested.

Once the UE and the network have successfully completed the primary authentication phase, the K_{AUSF} is derived by the Mobile Equipment (ME) and Authentication Repository and Policy Function (ARPF) from CK and IK during the 5G Authentication and Key Agreement (5G AKA) process. According to the Extensible Authentication Protocol (EAP) AKA' specification, K_{AUSF} is derived by the ME and AUSF, provided that the 3GPP credential K is used for authentication over a radio access technology supporting the EAP [12].

The anchor key K_{SEAF} is derived by the AUSF and ME from K_{AUSF}. The ME and Security Edge Access Function (SEAF) then use K_{SEAF} to derive K_{AMF}. The 3GPP standard delineates two types of access: trusted (3GPP access) and untrusted (non-3GPP access). The Non-3GPP Interworking Function (N3IWF) serves as a gateway for the 5GC, supporting the N2 and N3 interfaces towards the 5GC. Non-3GPP access encompasses access from various sources such as WiFi, WiMAX, fixed, and Code Division Multiple Access (CDMA) networks. The N3IWF key (K_{N3IWF}) is derived from K_{AMF} for non-3GPP access [12].

For the sake of protecting NAS signaling, the integrity and confidentiality keys, $K_{NAS_{int}}$ and $K_{NAS_{enc}}$, respectively, are derived by the ME and AMF from K_{AMF}. Moreover, K_{gNB} is derived by the ME and AMF from K_{AMF}. On the other hand, from K_{gNB} are computed the integrity and confidentiality keys for Access Stratum (AS), namely UP ($K_{UP_{int}}$ and $K_{UP_{enc}}$) and RRC ($K_{RRC_{int}}$ and $K_{RRC_{enc}}$), respectively [12].

However, the work in [13] points out that these measures alone may not provide comprehensive protection against specific types of fraud, such as fraudulent Update Location requests. Moreover, the evolving tactics exploited by hackers to evade security measures have prompted an increasing adoption of Artificial Intelligence (AI) solutions. These approaches aim to

effectively address the dynamic nature of suspicious behaviors and zero-day threats [13].

1.2.5 5G Challenges for Law Enforcement

The advent of 5G technology presents a multitude of new challenges for law enforcement, particularly in the realm of Lawful Interception [14]. These challenges, identified by Europol in 2019 [15], [16], concern the identification and localization, as well as the accessibility and availability of crucial information. These documents also highlight the fact that: *"it is critical that all these issues be addressed. More generally, it would be important for the European Union to discuss and take a comprehensive approach on all dimensions of 5G"* [16].

One of the primary challenges arises from the encryption of the International IMSI in 5G networks. This encryption complicates the task of security authorities in locating or identifying mobile devices and linking them to specific individuals. Moreover, advancements in 5G technology aim to render IMSI catchers obsolete, further complicating surveillance efforts.

Secondly, the implementation of network slicing in 5G networks introduces challenges in accessing information for lawful interception. Therefore, network slicing divides a single mobile radio network into multiple virtual networks tailored to specific applications or services, and this fragmentation may limit law enforcement's access to critical information, particularly in private slices held by unregulated entities.

In addition, Multi-Access Edge Computing (MEC) enables direct communication between terminal devices, bypassing the MO's core network. While this enhances response times, it complicates data retrieval for law enforcement by circumventing traditional communication routes [15].

Furthermore, the potential incorporation of End-to-End Encryption (E2EE) protocols in 5G standards poses a significant hurdle to lawful interception, as it is further explained in Section 1.3.

Finally, the virtualization of physical network components, that is NFV, presents additional obstacles for law enforcement. NFV undermines existing security measures aimed at protecting surveillance confidentiality, potentially enabling unauthorized access to monitored telephone numbers. Additionally, functions performed in one country can now be relocated abroad, necessitating the transfer of monitoring lists across jurisdictions and raising concerns about the confidentiality and integrity of law enforcement information [15].

1.3 END-TO-END ENCRYPTION

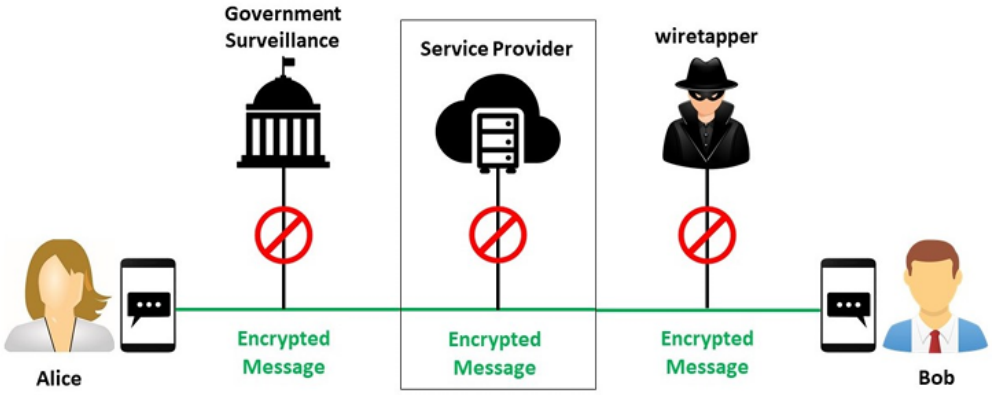


Figure 1.9: High-level diagram of the end-to-end encryption [17].

The rise of Instant Messaging (IM) and VoIP applications such as WhatsApp, Zoom, and Skype has led to a heavy reliance on online communications for audio, video, and text conversations. As these platforms are increasingly used for sharing sensitive information, concerns about government and law enforcement surveillance have arisen. Consequently, there is a growing need for secure methods to ensure confidentiality [17]. End-to-end encryption (E2EE) systems provide a solution by ensuring that only intended recipients can decrypt messages, without relying on intermediaries such as online services or centralized infrastructures like Public Key Infrastructure (PKI). This prevents service providers, government surveillance programs, or Man-in-the-Middle (MITM) attackers from reading the exchanged messages. In the realm of E2EE algorithms, the Off-the-Record (OTR) protocol proposed in 2004 aimed to provide secure communication [17]. Open Whisper Systems' Signal protocol, introduced in 2013, offers advanced security features such as forward secrecy and future secrecy, used by various IM and VoIP apps including Signal and WhatsApp. Google and Zoom have also implemented E2EE in their respective messaging platforms [17]. Secure end-to-end solutions provide the following security and privacy properties:

- **Confidentiality**: only the sender and recipient can access the shared contents.
- **Integrity**: messages cannot be altered while in transit.
- **Authentication**: recipients can verify the sender's identity.
- **Deniability, Forward, and Future Secrecy**: deniability ensures that neither message recipients nor anyone else with access to the conversation transcript can cryptographically prove the authorship of a specific message. Forward and Future Secrecy respectively protect the confidentiality of messages sent before and after an end-user device gets compromised.

Encryption entails costs, including technical expenses and significant engineering efforts. Law enforcement often opposes robust encryption, impeding access to suspects' communications. Moreover, encryption can obstruct user protection measures, especially on platforms reliant on content moderation for user safety. However, E2EE challenges traditional content moderation, limiting platform access to message contents, despite safeguarding privacy, data security, and human rights [18]. Consequently, addressing this trade-off necessitates the advancement of robust decryption capabilities and the establishment of cooperative frameworks between telecommunication service providers and LEAs [17].

1.4 KEY ESCROW

The concept of key escrow is gaining attention in the cryptographic field as a response to end-to-end encryption (E2EE) [19].

Key escrow emerged as a proposed solution to balance the need for privacy and security with the requirements of law enforcement and national security, which has been depicted in Section 1.3. Over the years, several key escrow solutions have been proposed and implemented. One of the first examples of key escrow systems was developed by the U.S. government in the early 1990s: the Clipper Chip [20]. It was a hardware encryption device that incorporated key escrow capabilities. Each Clipper Chip contained a unique cryptographic key split into two parts, with one part held by the user and the other by the government's escrow agents. This system allowed law enforcement agencies to access encrypted communications by obtaining the escrowed key. However, the Clipper Chip was abandoned in 1996 when it faced significant criticism due to security concerns, since the reliance on a government-held escrowed key raised fears of potential misuse or unauthorized access, undermining the privacy and security of encrypted communications.

In the recently adopted solutions, the Key Escrow Encryption System is designed to enable authorized individuals, including users and government officials, to decrypt encrypted data under specific circumstances [19]. This system, also known as key recovery, exceptional access, or data recovery, incorporates backup decryption capabilities. It employs designated *escrow agents* tasked with recovering encrypted communication sessions or files. These agents own the keys necessary for decryption, allowing them to decrypt data when required and, above all, when and until they are authorized by a proper warrant. In general, according to the schema proposed by the contribution in [19], a key escrow system comprises three main components:

1. **User Component:** this element, whether in software or hardware form, handles data encryption and decryption. It also securely transfers the secret key to the key escrow part.
2. **Key Escrow Component:** managed by the key escrow agent, this module oversees the storage, distribution, and utilization of data recovery keys.
3. **Data Recovery Component:** this component encompasses algorithms, protocols, and equipment essential for recovering plaintext information from ciphertext. It retrieves ciphertext using the information provided by the key escrow agent.

It's crucial to note that the system operates under the assumption of a full Trusted Third-Party (TTP). Moreover, key recovery poses risks by compromising encryption (potentially leading to improper disclosures of keys), introducing complexity, and incurring high costs. This undermines security guarantees, complicates management, and lacks a viable economic model, making it resource-intensive. These risks give rise to several concerns [21]:

- **Insider Abuse**: key recovery systems are vulnerable to compromise by authorized individuals, potentially leading to the abuse of authority and the disclosure of sensitive information.
- **New Targets for Attack**: key recovery agents become valuable targets for attackers due to their centralized collections of decryption keys, posing risks of data theft or spoofing of encryption controls.
- **Forward Secrecy**: key recovery undermines forward secrecy, making encrypted communications vulnerable to disclosure long after they occur, as the keys must be stored instead of destroyed.

Moreover, the scale and operational complexity of key recovery infrastructure pose significant challenges, including the need for extensive coordination among thousands of entities worldwide. This complexity increases the likelihood of operational vulnerabilities, fraudulent key requests,

and compromised security. In this project, we proceed with the assumption that one or more TTPs are in position and have established a suitable infrastructure for receiving and managing all keys deposited by their registered users [21].

2 THE PROPOSED LI FRAMEWORK

This chapter analyzes the methodology and the software presented by the authors in the contribution [14], which have been adopted for the implementation of an experimental Lawful Interception infrastructure in 5G networks, in accordance with the theoretical concepts and standards outlined in the previous chapter. The architecture is designed to enable interaction between two users (UEs) connected to the same 5G network but each to a different gNB, in two different scenarios:

1. The exchange of end-to-end encrypted packets containing data files (e.g., text messages, images, videos, etc.) using the connection-oriented TCP protocol.

2. A Voice over IP (VoIP) call between the two users, encrypted using the Secure Real-time Transport Protocol (SRTP)/Session Description Protocol Security Descriptions (SDES) protocol.

Furthermore, as described in the standard introduced in Section 1.1, during the packet exchange, the Lawful Interception infrastructure combined with an appropriate Key Escrow algorithm enables the Law Enforcement Agency (LEA) to access intercepted packets and successfully decrypt them.

2.1 IMPLEMENTATION ENVIRONMENT AND TOOLS

In this section, the different environments that constitute the proposed 5G Lawful Interception framework, developed in this thesis, are introduced: the 5GC, the 5G NR, the Lawful Interception environment, and the VoIP system. Initially, the specific docker network architectures and the software used for each of them are presented. Subsequently, the discussion and analysis focus on the libraries and implementation methods for managing the transmission of data streams, along with the key escrow algorithm used. An overview of the software and the tools utilized in the project is presented in Table 2.1.

Table 2.1: List of software and tools [14].

5G Network and Lawful Interception	**Software**
Access Network	UERANSIM
5G Core Network	Open5gs
Lawful Interception	OpenLI
End-to-end communication	Netcat
VoIP services	Asterisk Server and PJSIP library
Cryptographic Operation	**Adopted libraries**
Hash function	Hashlib, libnum
Key derivation function	PyCryptodome
Encryption, decryption	PyCryptodome
Bilinear pairing	Tate_bilinear_pairing
Post-processing step	**Software**
Interception	OpenLI and libtrace
Packet decapsulation	Scapy
Reassembly	TCPReassembly

2.1.1 5G Core Network - Open5GS

Open5GS [22] is an open-source project that offers a software implementation of a comprehensive 5G network. It encompasses crucial functionali-

ties such as user session control, mobility, network management, security, and more. Developed as part of the open-source 5G ecosystem, Open5GS facilitates the experimentation, testing, and deployment of 5G networks on standard hardware and virtualized infrastructure.

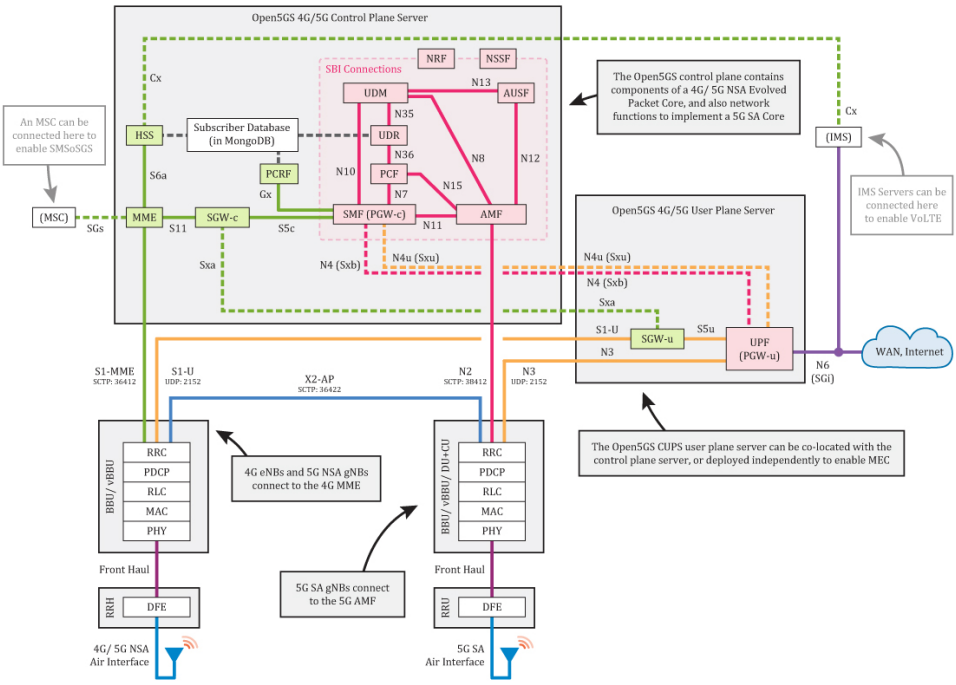


Figure 2.1: Open5GS architecture [22].

Open5GS software currently supports 3GPP Release 17 and provides the Network Functions (NFs) of the 5G Core (5GC) network for both Standalone (SA) and non-SA network development. In this thesis, our focus lies on replicating only the SA 5GC, which incorporates the following NFs (as previously described in Section 1.2.1):

- Network Repository Function (NRF);
- Service Communication Proxy (SCP);
- Access and Mobility Management Function (AMF);
- Session Management Function (SMF);
- User Plane Function (UPF);

- Authentication Server Function (AUSF);
- Unified Data Management (UDM);
- Unified Data Repository (UDR);
- Policy and Charging Function (PCF);
- Network Slice Selection Function (NSSF);
- Binding Support Function (BSF).

The complete Open5GS architecture is illustrated in Figure 2.1. To initiate the Docker implementation of Open5GS, we utilize the *herlesupreeth* repository [23], which facilitates the creation of containers for the 5G Core Network.

The usage of a Dockerized version for the core network offers several advantages. Docker [24] is an open-source containerisation platform that allows applications and services to be created, deployed and managed in lightweight, isolated containers. Compared to virtual machines, Docker offers advantages such as portability, isolation, simplified management, reproducibility, scalability and security features. Docker Compose, used extensively in our framework, allows Docker multi-container applications to be defined and executed using a *yaml* file to configure the application services.

Provided that the containers' images have already been built with *docker compose build {image}*, the single command *docker compose up* creates and initiates all the services specified in the *docker-compose.yaml* file. This approach simplifies application lifecycle management and facilitates communication between containers and external networks, as depicted in Figure 2.2.

The repository's *.env* file specifies the IP addresses of various containers and networks (as shown in Figure 2.2). By changing these parameters we can modify the IP addresses of the containers created or adjust the parameters for simulated users. Within the *open5gs* folder, the file

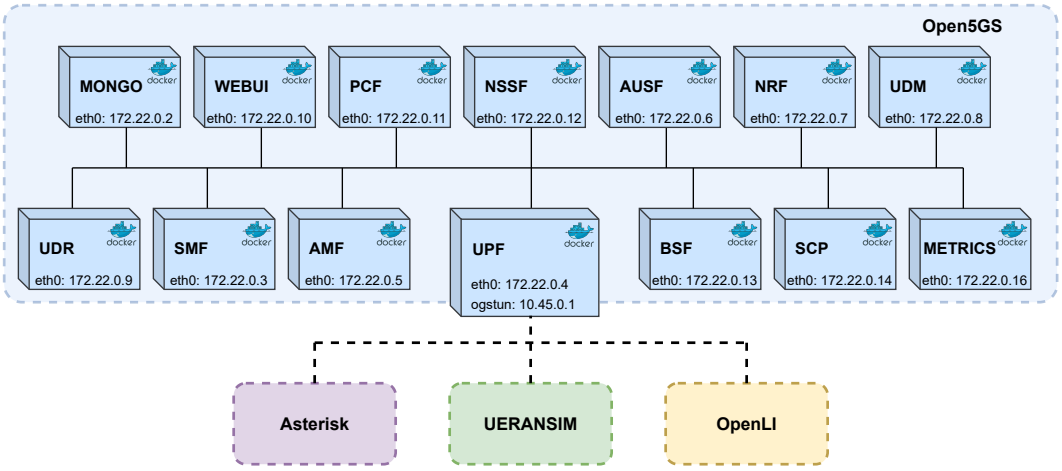


Figure 2.2: Open5GS containers.

docker-compose.yaml is useful for starting the listed containers, exposing the necessary ports. The Web User Interface (UI) container, accessible at *http://localhost:3000*, offers a graphical interface for 5G network configuration and user registration. The default login credentials are username: *admin* and password: *1423*. This interface enables uploading user data for later connection via UERANSIM, storing it in a separate MongoDB database container. Figure 2.3 illustrates the Graphical User Interface (GUI) provided by this container and the parameters for the first user, allowing adjustments to various settings from its interface:

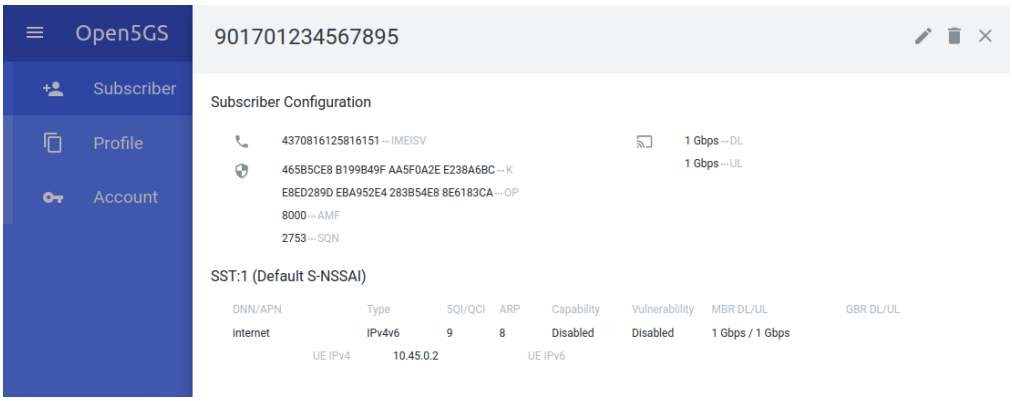


Figure 2.3: UE_1's information configured through WebUI interface at *http://localhost:3000*.

- **International Mobile Subscriber Identity (IMSI)**: this is the unique user identifier known as Subscriber Concealed Identifier (SUCI) in 5G

networks, consisting of a series of numbers that uniquely identify a subscriber in a mobile network. IMSI typically follows this structure:

  - **Mobile Country Code (MCC)**: a three-digit code that uniquely identifies the country in which the subscriber is registered. For example, "310" for the US, "222" for Italy and "901" for testing purposes.
  - **Mobile Network Code (MNC)**: a code consisting of two or three digits identifying the operator or mobile network in the country. For instance, "01" is the MNC code of the operator TIM in Italy.
  - **Mobile Subscriber Identification Number (MSIN)**: it is the part of the IMSI which uniquely represents the subscriber within his network and operator.

  Therefore, the full format of the IMSI is MCC|MNC|MSIN. For instance, in this thesis work the IMSI configured for the first subscriber is "901701234567895".

- **Subscriber Key (K)**: it is used in the mutual authentication process and session key negotiation to encrypt and decrypt data between the user and the network.

- **Authentication Management Field (AMF)**: value used for security within the 5G network and part of the authentication process, AMF protects the user from attacks like illegal data interception.

- **Operator Key (OPc/OP)**: it is a cryptographic key utilized in mobile networks for authentication and security purposes.

- **UE IPv4 Address**: this is the IP address assigned to a user device in mobile networks when connected to the Internet. For simplicity, in this project a static IP is assigned to ensure consistency for a user with a given IMSI.

After launching the Open5GS containers and configuring the parameters via the web interface, it is possible to launch the UERANSIM containers for gNBs and UEs and attach them to the 5GC.

2.1.2 5G New Radio - UERANSIM

UERANSIM [25] is a cutting-edge open-source simulator for 5G Standalone (SA) gNBs and UEs. This project is handy for testing 5G Core Network and exploring the 5G system.

Using UERANSIM, containers are configured and launched to act as gNBs and UEs within the 5G architecture developed for simulations within this study. The gNBs, along with their respective UEs, form what is known as the Radio Access Network in the 5G architecture. This access interface interacts with the 5GC system, as described in the preceding section, to manage network attachment procedures and exchange control packets (involving, among others, AMF, SMF, AUSF, etc.), as well as to generate a data stream (routed through the UPF), as illustrated in Figure 2.4.

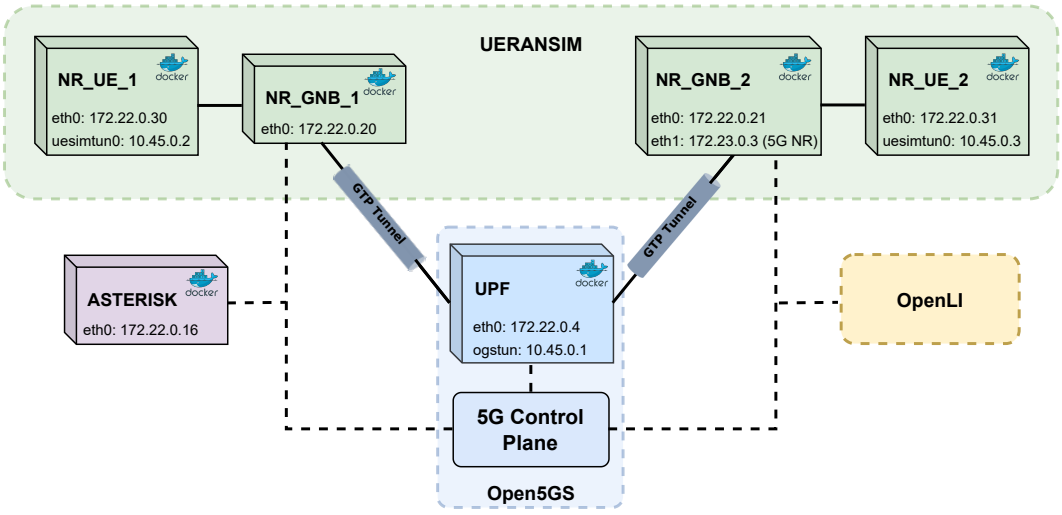


Figure 2.4: Architecture of the deployed 5G NR via UERANSIM containers and their connections with the 5GC to enable the two UEs to securely communicate.

With regard to our deployment analysis, within the *herlesupreeth* [23] repository, the implementation requires adjustments to the *nr-gnb.yaml* and *nr-ue.yaml* files. These changes facilitate the creation of a new gNB and an additional user, establishing a scenario where two users are connected to two distinct gNBs, everything integrated into the same Core Network. The main codes for configuring the UERANSIM environment are:

- ***ueransim_image_init.sh***: this script is executed after the command *docker-compose -f nr-gnb.yaml up* or *docker-compose -f nr-ue.yaml up*. It specifies actions to be taken based on the name of the container being run. The script is adjusted to pass the gNB's or UE's identification number to the next two files described below.

- ***open5gs_gnb_init.sh***: this file is invoked from the previous one and, with the identifier passed as input, assigns the IP of gNB_1 or gNB_2 depending on the container. Additionally, it configures the containers regarding the gNBs by passing parameters such as MCC, MNC, the gNB's IP, and the AMF's IP.

- ***open5gs_ue_init.sh***: this file, invoked by *ueransim_image_init.sh*, assigns parameters such as International Mobile Subscriber Identity (IMSI), Subscriber Key (K), Authentication Management Field (AMF), Operator Key (OPc/OP), International Mobile Equipment Identity (IMEI), and gNB's IP. These parameters vary based on the container identifier (whether it is UE_1 or UE_2), and for the authentication process to be successful, the first four parameters listed must match. The IP of the gNB is then used to connect the UE to the network by selecting one of the two configured gNBs.

To complete the network configuration, the routing rules of the UE containers must be modified. Typically, these containers are able to communicate via the interface *uesimtun* (which connects them to the 5G Core Network) only through the *nr-binder* script. To enable any UE message to pass

through the core network, the IP assigned to the *uesimtun* interface is set as the default route.

2.1.3 Lawful Interception - OpenLI

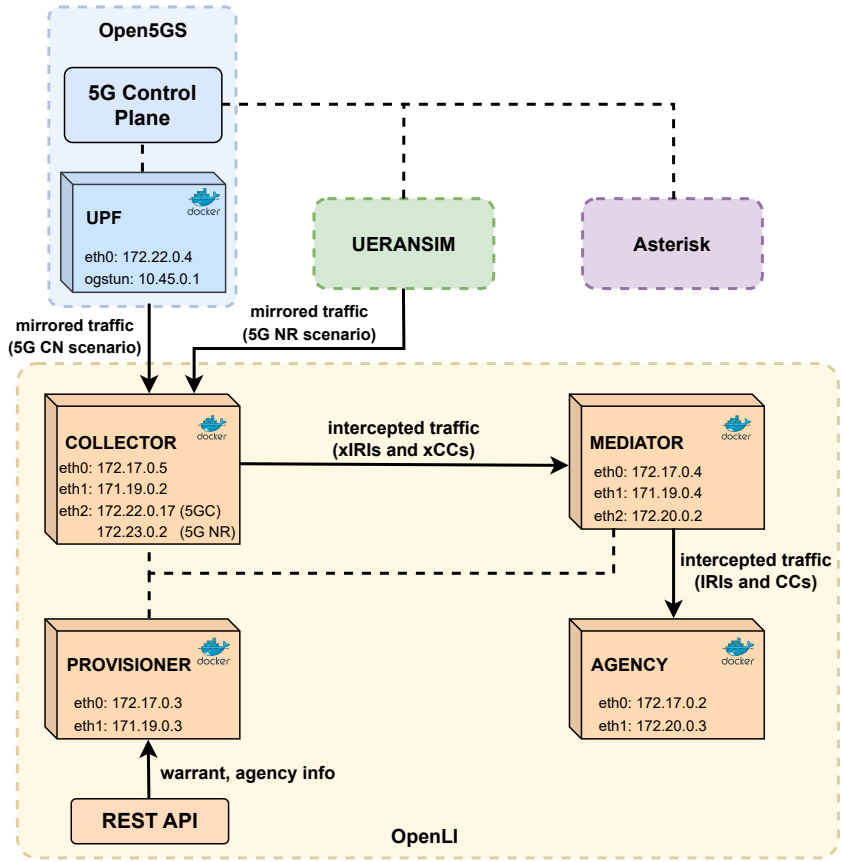


Figure 2.5: OpenLI containers with regards to the interception workflow when the interception occurs either at UPF or at gNB. The network *docker*0, attached to the containers' interface *eth*0, is not displayed in the connections since it is not relevant for the proposed LI framework.

OpenLI [26], developed by the WAND Network Research Group at the University of Waikato, is a software solution compliant with the ETSI standards for Lawful Interception of IP and other protocols. The software, constructed on the *libtrace* packet processing library, is distributed under the GPLv3 license.

OpenLI captures packets directly from the network and formats them for transmission to law enforcement agencies, without attempting to decrypt or inspect the content of the packets beyond the TCP or UDP headers, except for Session Initiation Protocol (SIP) and Remote Authentication Dial-In User Service (RADIUS) packets, which are fully analyzed in order to identify relevant packets to be intercepted. The OpenLI software consists of three main components:

- **The Provisioner** is the central controller for the entire OpenLI system. Once deployed and operational, it becomes the reference point for interaction with OpenLI via the REpresentational State Transfer (REST) API interface, which is used to interact with the OpenLI software, thus it should be configured to restrict access only to the authorized users. In detail, the REST API is useful for transmitting configuration commands to register the LEA and receive warrants, and then the provisioner is able to initiate and terminate packet capturing in response to those requests of interception. Unauthorized access could potentially compromise ongoing intercepts or facilitate unauthorized interception activities.

- **The Collector** is a crucial component within OpenLI, since it is responsible for intercepting network traffic and then bears the highest computational load. Packet capture is integral to collector operations, and selecting an appropriate capture method is essential to handle traffic loads efficiently. While collectors support packet capture on multiple interfaces, efforts should be made to minimize extraneous traffic to reduce CPU load. Collectors also manage session and IP tracking, primarily through protocols like RADIUS. This enables them to determine whether a packet should be intercepted based on the user's intercept status. Upon interception, packets are encoded according to ETSI standards, which is a computationally intensive task. To meet these standards, which impose additional complexi-

ties such as ensuring consistent sequence numbering and identifiers across intercepts, OpenLI developed *libwandder* for encoding and decoding records. Finally, collectors forward encoded records to mediators, ensuring also proper sequencing and temporary RabbitMQ buffering to account for mediator unavailability. Distributing multiple collector instances throughout the network is a key design feature, enhancing scalability and ensuring comprehensive interception coverage. However, careful consideration of hardware specifications is necessary to prevent packet loss and incomplete intercepts, which could impact the law enforcement agency QoS.

- **The Mediator** serves as a pivotal component acting as an intermediary between the Collector and the Law Enforcement Agencies (LEAs), performing a range of critical tasks simultaneously. Firstly, it maintains a comprehensive record of active intercepts and the respective LEAs that have requested them, all of which are provided by the provisioner. Secondly, the mediator establishes and sustains TCP sessions for both HI2 and HI3 interfaces with each designated LEA. This continuous connectivity ensures that LEAs remain informed about the operational status of the LI system. Lastly, the mediator's core responsibility entails receiving encoded ETSI records from collectors and directing them to the appropriate LEA via the corresponding interfaces. In instances where an interface is temporarily unavailable, the mediator stores them in a RabbitMQ buffer until the communication becomes available again.

Furthermore, a crucial aspect of the deployment process involves conducting tests directly with the LEAs. In reality, the equipment used by LEAs for receiving and decoding intercepts can often be more expensive than commercial products for performing intercepts. However, for the purposes of this project, we choose the **Agency** container provided by OpenLI, which allows for simple and ETSI-compliant emulation of packet reception

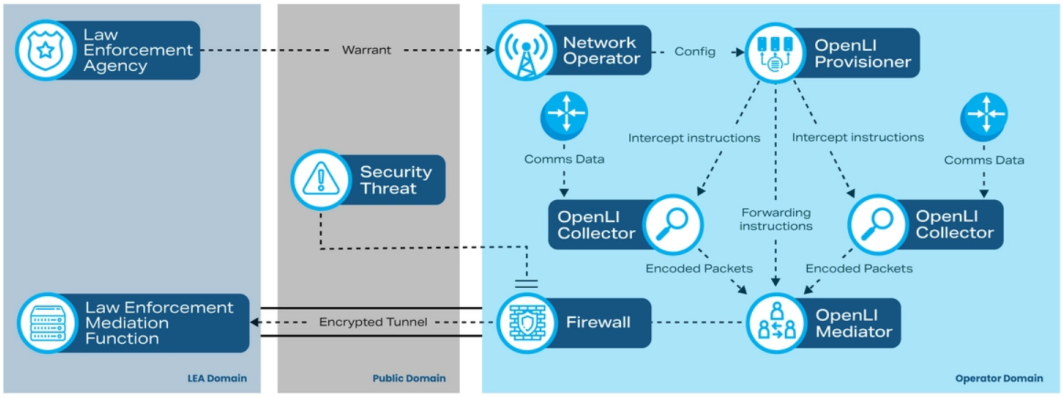


Figure 2.6: OpenLI overall architecture [26].

by a LEA. Indeed, the *libtrace* library is capable of receiving and parsing ETSI sessions, enabling any *libtrace* tool or program to emulate a LEA and serve as a receiver for mediator transmissions. Our emulated LEA listens for connections from the mediator, requiring knowledge of the container's IP address to establish a listening socket on the correct interface. Each agency connection requires its TCP port number, with HI2 for IRI records and HI3 for CC records. Through the REST API, the provisioner can be properly configured to recognize our agency's existence and authorize the transmission of IRI and CC records.

Within the proposed LI framework, each of the instances provided by OpenLI for conducting ETSI-compliant interceptions corresponds to an appropriate Docker image, which is launched during the setup phase of the simulation environment and configured accordingly with its network interfaces and specific parameters.

2.1.4 End-to-end communication libraries

As presented in Section 2.1.2, UERANSIM enables two User Equipments (UEs), connected to the same 5G network, to establish PDU sessions and exchange packets via the *uesimtun* network interface, which models GTP tunneling typically configured between gNBs and UPF in mobile radio networks. The *nr-binder* script facilitates the execution of commands and ap-

plications by binding them to the *uesimtun* interface. In general, the syntax is as follows:

```
./nr-binder {PDU-SESSION-IP-ADDRESS} {COMMAND} {ARGS}
```

For example, if one wishes to execute a ping from UE_1 with IP address 10.45.0.2 to UE_2 at 10.45.0.3, the command syntax would be:

```
./nr-binder 10.45.0.2 ping 10.45.0.3
```

In the proposed LI framework, various applications and software are used to implement encrypted file transfer and VoIP call with SRTP/SDES, which are going to be presented in the following subsections.

Netcat

Specifically, for establishing a TCP connection for file transfer, the *netcat* tool was used. Netcat, commonly abbreviated as *nc*, serves as a computer networking utility facilitating reading from and writing to network connections via TCP or UDP. Positioned as a reliable backend, it is adaptable for direct use or seamless integration into various programs and scripts. Additionally, it serves as a comprehensive network debugging and investigative tool, offering a wide array of connection capabilities and built-in functionalities, including port scanning, file transferring, and port listening.

For the sake of clarity, let us consider the general syntax of Netcat for transferring an image between two hosts. To begin, on the receiving end, initiate a Netcat listener on a specified port to receive the image:

```
nc -l -p <receiver_port> > received_image.png
```

Next, on the sending end, use Netcat to send the image to the receiver's IP address and port number:

```
nc <receiver_ip> <receiver_port> < image.png
```

Replace *nc <receiver_ip>* with the IP address of the receiver (e.g. the UE_2 at 10.45.0.3) and *<receiver_port>* with the corresponding port number.

Asterisk Server

The Asterisk Server [27] is an open-source software designed for managing telephone calls, offering advanced functionalities like call transfers and conference calls through both VoIP and traditional networks. It provides customization options to tailor telephone exchanges according to the specific requirements of businesses and organizations. Asterisk is particularly essential for implementing telephony over IP services. With its configuration, it becomes possible to establish a private exchange, defining the registered users and the configurations to be applied.

During the creation of the Docker image, a series of commands are executed, including the installation of necessary packages, the download of the Asterisk archive from its official source, and the creation of a directory for Asterisk keys. Then, configuration and script files are copied to the Docker image directory. Specifically, Asterisk configuration files and related files are placed in the */etc/asterisk* directory within the Docker image. The ports exposed by the Docker image are specified, and they are ports 5060, 5061, 5070, which are commonly used in Asterisk configurations for VoIP communication. Ports 5060 and 5061 are associated with the *pjsip* driver, utilizing UDP and Transport Layer Security (TLS) protocols, respectively.

The **pjsip** driver, integral to Asterisk, facilitates SIP-based VoIP functionality within its telephony system. Unlike the PJSIP library, which is an independent open-source software library, the *pjsip.conf* file configures the Asterisk Server for user registration and call handling. This configuration is generally structured into two primary sections:

- **Transport protocols definition**: transport protocols like UDP and TLS are defined, along with their listening ports. TLS is particularly important for encryption in systems like SRTP/SDES, requiring specifications such as TLS version (v1.2), the server's private key, and TLS handshake certificate.

- **User definition**: users, identified by telephone numbers (e.g., 101 and 102), are detailed here. Each user's setup includes three subsections:
  - Endpoint: specifies call handling, supported codecs (e.g., A-law), and media encryption protocols like SRTP/SDES.
  - Auth: specifies authentication methods (e.g., username and password) and credentials.
  - Address of Record (AOR): limits each user to one registered contact.

During calls, the Asterisk Server acts as a proxy, decrypting incoming SRTP traffic from one user and forwarding it to another based on SIP message keys. SIP message encryption occurs over TLS, while the SRTP stream uses UDP for transport and it is encrypted end-to-end using SRTP/SDES.

PJPROJECT

PJPROJECT [28], written in C language, is an open-source multimedia communication library. It implements standard protocols combining the signaling protocol SIP with a robust multimedia framework and Network Address Translation (NAT) traversal functionality. This integration forms a high-level API that's portable and adaptable to various systems, from desktops to embedded systems and mobile handsets. Compact and feature-rich, PJPROJECT supports audio, video, presence, and instant messaging, complemented by extensive documentation.

As it is shown in Figure 2.7, the PJPROJECT architecture offers several libraries that can be grouped into three main categories:

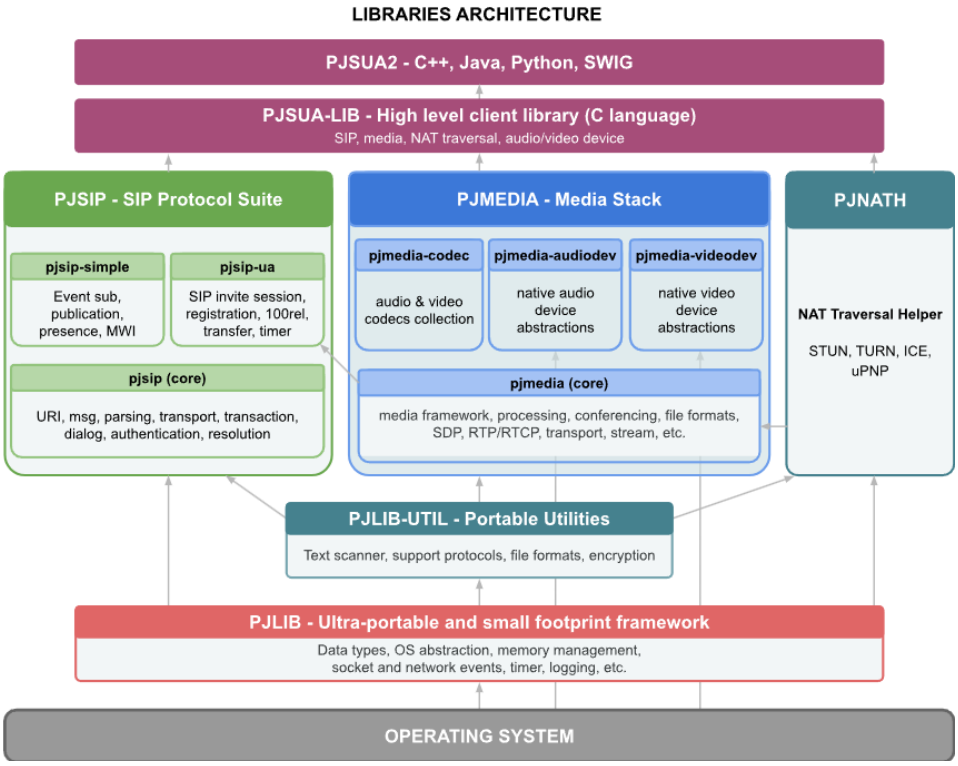


Figure 2.7: PJPROJECT architecture [28].

- **PJSIP** is small-footprint and high-performance SIP stack written in C, distributed under GNU General Public License (GPL), which supports a set of features and extensions of the SIP protocol.
- **PJMEDIA** is a specialized library focused on managing multimedia data transfer. It encompasses features like managing Real-time Transport Protocol (RTP)/Real-time Transport Control Protocol (RTCP) stacks, supporting also Secure Real-time Transport Protocol (SRTP) through the *libSRTP* library.
- **PJNATH** is an open source library providing NAT traversal functionalities using standard based protocols such as uPNP, STUN, TURN, and ICE.

Furthermore, **PJLIB** and **PJLIB-UTIL** serve as foundational support libraries. PJLIB abstracts system-dependent functionalities, ensuring basic operations across different operating systems. On the other hand, PJLIB-UTIL builds upon PJLIB, offering more complex functionalities, particu-

larly in cryptography, including algorithms like SHA1, MD5, HMAC, and CRC32. These libraries provide the necessary groundwork for higher-level functionalities in the PJSUA2 framework.

PJSUA2 API is a C++ library on top of PJSUA-LIB API to provide high level API for constructing SIP multimedia user agent applications. Using the PJSUA2 API, developers can easily build SIP-based communication applications. In our project, we use the PJSUA2 Python wrapper to develop scripts enabling SIP session establishment between two UEs and, consequently, facilitating a VoIP call supervised by the Asterisk Server.

Utilities for packet decapsulation and reassembly

Two libraries are included in the proposed LI framework in order to enable GTP packet management and, with precise regards to the end-to-end file transferring, to allow the reconstruction of the transmitted file from the received TCP flow:

- **Scapy** is an interactive packet manipulation tool capable of crafting and interpreting packets across a wide range of protocols. It offers both command-line interaction and a Python library interface. By utilizing Scapy's Python library, custom scripts process captured packets. Scapy's processing capabilities facilitate the management of GTP-U packets captured by *tracelib*, enabling the creation of scripts to categorize these packets based on the application protocol utilized. Additionally, Scapy is used to decapsulate the intercepted GTP packets, allowing for the extraction of the TCP or SRTP payloads which are relevant to file and VoIP interceptions, respectively. This enables the transmission of the intercepted data to the LEA.

- **PcapPlusPlus** is a portable C++ library designed for capturing and analyzing network traffic. While PcapPlusPlus offers a broad range of functionalities, it is used in the proposed LI framework for its support of community-built applications, including TcpReassembly.

- **TcpReassembly** is a command-line utility packaged with PcapPlusPlus, which processes packets stored in a *pcap* file and generates various *txt* files containing reconstructed session payloads. Each file contains the exchanged application-level data during the session, and this process allows both the UEs and the LEA to reconstruct the encrypted data flow.

When it comes to reassembling the SRTP stream, this is seamlessly handled by the PJSUA2 library, which relies on the PJMEDIA library. Similar to TcpReassembly, PJMEDIA is able to convert a *pcap* file into a media file, that is still encrypted according to the SRTP protocol.

2.1.5 End-to-End encryption and Key Escrow

The ID-based cryptosystem (IDBC) encryption algorithm presented in [29], whose implementation is further examined in the Appendix, manages the Key Escrow procedure. Briefly, it allows on one hand the generation and exchange of the session key between the two UEs, and on the other hand enables the Escrow Agent (i.e., the LEA) securely and circumstantially to retrieve the same key k_{AB}. In order to implement the IDBC scheme in our project, the following cryptographic libraries are used:

- **Libnum** provides essential mathematical tools for cryptographic operations in Python, including prime management and modular arithmetic.

- **Hashlib** ensures data integrity by generating secure hash values for IDs, using various cryptographic hash algorithms.

- The **tate_bilinear_pairing** library facilitates cryptographic operations involving elliptic curves, enabling point addition and scalar multiplication which are essential for secure computations.

- **PyCryptodome** serves as a comprehensive Python package offering robust cryptographic primitives, utilized for implementing advanced encryption systems like AES-256 and key derivation functions.

2.2 TESTBED AND NETWORK CONFIGURATION

After introducing the software and the development methodology used to create the proposed Lawful Interception framework, we proceed to discuss the actual deployment of the architecture in [14]. In the following sections, we illustrate the configuration phase of the connected entities in the network, covering both the 5G architecture and the LI architecture. The correctness of these configurations can be verified through system logs. Subsequently, once a functional and operational environment is established, we analyze the interception phases and how they are effectively managed by the proposed LI framework.

In this section we consider the scenario where the interception is carried out by a Point of Interception (POI) located in the Core of the 5G network, specifically at the User Plane Function (UPF). The slight modifications made to the setup phase concerning interception with a POI positioned at the network edge, are detailed in the dedicated Section 2.4.

To facilitate the configuration of variables such as IP addresses for the interfaces of individual containers, which are also useful for creating different networks, they are set into the *.env* file.

For the initialization of the entire network system, it is necessary to simply execute the script *start_all.sh*. This bash code is responsible for running each container, loading and installing the required libraries, and orchestrating the setup of the network. The bootstrapping phase of the proposed LI framework is composed of three primary stages, which are further described in the subsequent sections:

1. Startup of 5GC (Open5GS) and 5G NR (UERANSIM).

2. Establishing the VoIP system for handling calls (Asterisk Server).

3. Initiating the environment for Lawful Interception (OpenLI).

Before proceeding, it is essential to introduce a highly simplified version of the directory tree that constitutes the *5GLI_Framework* project:

Each subfolder, as implied by its name, contains scripts and/or configuration files organized according to their scopes: the *open5gs* directory encompasses materials for setting up the 5G network, while the *openli-training-lab* directory pertains to the Lawful Interception environment, and *pythKeyEscrow* houses the implementation of the end-to-end encryption algorithm. Lastly, as demonstrated in the sections concerning simulation exe-

cution, the *script* folder contains all main execution scripts (e.g. *start_all.sh* to initialize the system) for the various phases of the scenarios implemented within the proposed framework. Before going into the detail of each subsystem, a broad overview of the networks configured to facilitate communication among the various entities operating within the proposed architecture is provided in Table 2.2.

2.2.1 5G Network Initialization and VoIP setup

As previously highlighted, the information required to launch and configure the containers related to 5GC and 5G NR is contained within specific *yaml* files utilized by *docker compose*. Indeed, the *start_all.sh* script executes:

```
# working in open5gs directory
set -a
source .env
sudo docker compose up -d  # to run 5GC containers
sudo docker compose -f nr-gnb.yaml up -d
sudo docker compose -f nr-ue.yaml up -d
```

Following these commands, and as already seen in sections 2.1.1 and 2.1.2, after configuring the WebUI and executing the init scripts for gNBs and UEs, it is possible to verify the correctness of the configuration and execution of each entity by inspecting the containers' log files. For the sake of clarity, we provide only the relevant logs pertaining to AMF and UPF for the 5GC, and to gNB_1 and UE_1 for the 5G NR.

Table 2.2: Overall presentation of the networks deployed in the architecture.

Name	Network IP Address	Description
docker0	172.17.0.0/16	Default bridge docker network, connecting all the running containers (not relevant for the project).
open5gs_default	172.22.0.0/24	Network deployed to connect entities related to the 5GC and 5G NR, including the Asterisk server. This network is used for control messages and to enable VoIP proxy communication.
uesimtun0	10.45.0.0/24	Network used to establish the end-to-end tunnel between the UEs and the UPF.
openli-agency	172.20.0.0/24	Network that implements the HI2 and HI3 interfaces between the Mediator (MDF) and the Agency (LEMF).
openli-lab	172.19.0.0/24	Local bridge network enabling LI containers (provisioner, mediator, and collector) to exchange control messages along with xIRIs and xCCs.
openli-edge	172.23.0.0/24	Solely in the scenario where interception occurs within the 5G NR, this network is utilized to convey the mirrored the traffic from the gNB to the collector.

Code 2.1: AMF logs.

```
marco@marco:~$ sudo docker logs amf
    Deploying component: 'amf-1'
    Open5GS daemon v2.6.4-105-g0abfb20
    [app] INFO: Configuration: '/open5gs/install/etc/open5gs/amf.yaml'
    [app] INFO: File Logging: '/open5gs/install/var/log/open5gs/amf.log'
    [sbi] INFO: NF Service [namf-comm]
    [sbi] INFO: nghttp2_server() [http://172.22.0.5]:7777
    [amf] INFO: ngap_server() [172.22.0.5]:38412
    [sctp] INFO: AMF initialize...done
    [amf] INFO: gNB-N2 accepted[172.22.0.21]:50032 in ng-path module
    [amf] INFO: gNB-N2 accepted[172.22.0.21] in master_sm module
    [amf] INFO: [Added] Number of gNBs is now 1
    [amf] INFO: gNB-N2 accepted[172.22.0.20]:49039 in ng-path module
    [amf] INFO: [Added] Number of gNBs is now 2
    [amf] INFO: InitialUEMessage
    [amf] INFO: [Added] Number of gNB-UEs is now 1
    [amf] INFO: RAN_UE_NGAP_ID[1] AMF_UE_NGAP_ID[1] TAC[1] CellID[0x10]
    [amf] INFO: [Added] Number of AMF-UEs is now 1
    [gmm] INFO: Registration request
    [gmm] INFO: [suci-0-901-70-0000-0-0-1234567899] SUCI
    [amf] INFO: InitialUEMessage
    [amf] INFO: [Added] Number of gNB-UEs is now 2
    [amf] INFO: RAN_UE_NGAP_ID[1] AMF_UE_NGAP_ID[2] TAC[1] CellID[0x10]
    [amf] INFO: [Added] Number of AMF-UEs is now 2
    [gmm] INFO: Registration request
    [gmm] INFO: [suci-0-901-70-0000-0-0-1234567895] SUCI
    [gmm] INFO: [imsi-901701234567899] Registration complete
    [amf] INFO: [imsi-901701234567899] Configuration update command
    [amf] INFO: [Added] Number of AMF-Sessions is now 1
    [gmm] INFO: [imsi-901701234567895] Registration complete
    [amf] INFO: [imsi-901701234567895] Configuration update command
    [amf] INFO: [Added] Number of AMF-Sessions is now 2
```

Code 2.2: UPF logs.

```
marco@marco:~$ sudo docker logs upf
    Deploying component: 'upf-1'
    [app] INFO: Configuration: '/open5gs/install/etc/open5gs/upf.yaml'
    [app] INFO: File Logging: '/open5gs/install/var/log/open5gs/upf.log'
    [pfcp] INFO: pfcp_server() [172.22.0.4]:8805
    [pfcp] INFO: ogs_pfcp_connect() [172.22.0.3]:8805
```

```
    [gtp] INFO: gtp_server() [172.22.0.4]:2152
    [app] INFO: UPF initialize...done (../src/upf/app.c:31)
    [upf] INFO: PFCP associated [172.22.0.3]:8805
    [upf] INFO: [Added] Number of UPF-Sessions is now 1
    [gtp] INFO: gtp_connect() [172.22.0.3]:2152
    [upf] INFO: UE F-SEID[UP:0x753 CP:0xbf7] APN[internet] PDN-Type[1]
    ↪ IPv4[10.45.0.2] IPv6[]
    [upf] INFO: UE F-SEID[UP:0x753 CP:0xbf7] APN[internet] PDN-Type[1]
    ↪ IPv4[10.45.0.2] IPv6[]
    [gtp] INFO: gtp_connect() [172.22.0.21]:2152
    [upf] INFO: [Added] Number of UPF-Sessions is now 2
    [upf] INFO: UE F-SEID[UP:0xa0 CP:0x334] APN[internet] PDN-Type[1]
    ↪ IPv4[10.45.0.3] IPv6[]
    [upf] INFO: UE F-SEID[UP:0xa0 CP:0x334] APN[internet] PDN-Type[1]
    ↪ IPv4[10.45.0.3] IPv6[]
```

Code 2.3: gNB_1 logs.

```
marco@marco:~$ sudo docker logs nr_gnb_1
    Deploying component: 'ueransim-gnb-1'
    [sctp] [info] Trying to establish SCTP connection...
    ↪ (172.22.0.5:38412)
    [sctp] [info] SCTP connection established (172.22.0.5:38412)
    [sctp] [debug] SCTP association setup ascId[5]
    [ngap] [debug] Sending NG Setup Request
    [ngap] [debug] NG Setup Response received
    [ngap] [info] NG Setup procedure is successful
    [rrc] [debug] UE[1] new signal detected
    [rrc] [info] RRC Setup for UE[1]
    [ngap] [debug] Initial NAS message received from UE[1]
    [ngap] [debug] Initial Context Setup Request received
    [ngap] [info] PDU session resource(s) setup for UE[1] count[1]
```

Code 2.4: UE_1 logs.

```
marco@marco:~$ sudo docker logs nr_ue_1
Deploying component: 'ueransim-ue-1'
    [sctp] [info] Trying to establish SCTP connection...
    ↪ (172.22.0.20:36492)
    [sctp] [info] SCTP connection established (172.22.0.20:36492)
    [sctp] [debug] SCTP association setup ascId[5]
    [rrc] [debug] UE[1] new signal detected
    [rrc] [info] RRC Setup for UE[1]
```

```
[rrc] [debug] Random Access Request received
[rrc] [info] RRC Connection Setup complete, RNTI[1]
[rrc] [info] S-TMSI[0x00000000] is assigned to UE[1]
[rrc] [info] UE[1] is successfully registered
[nas] [debug] Initial Registration Request received
[nas] [info] Network Name[Open5GS]
[nas] [debug] Initial Registration accept sent
[nas] [info] UE[1] is attached to the network
[nas] [debug] Security Mode Command received
[nas] [debug] Selected integrity[1] ciphering[1]
[nas] [debug] Registration complete received
[nas] [info] UE[1] is registered with IMSI[901701234567890]
[nas] [info] Initial Registration is successful
[nas] [debug] PDU Session Establishment Request received
[nas] [debug] UAC access attempt is allowed for identity[0],
↪ category[MO_sig]
[nas] [info] PDU Session establishment is successful PSI[1]
[app] [info] Connection setup for PDU session[1] is successful, TUN
↪ interface[uesimtun0, 10.45.0.2] is up.
```

Regarding the initialization of the Asterisk server, this occurs exclusively through *docker compose*, as demonstrated in Section 2.1.4, which handles the configuration of *pjsip*. Consequently, packets are exchanged between the Asterisk server and the two UEs for the TLS handshaking process, which is a standardized procedure involving cipher suite negotiation, authentication through certificate exchange, and the negotiation of a symmetric key for the communication between the UE and the Asterisk Server, which is distinct from the one protecting the end-to-end intercepted communication.

Moreover, as the realization of the VoIP call requires both UEs to have access to the host's audio peripherals (i.e., the microphone for voice recording), it is imperative to execute this command at system's first startup:

```
1 pactl load-module module-native-protcol-tcp port=34567
    ↪ auth-ip-acl=172.22.0.0/24
```

where 172.22.0.0/24 represents the network address encompassing the UE, gNB, and Asterisk server. Indeed, within the context of PulseAudio (which

is an audio server for Unix-based systems), this command serves to load the module enabling audio communication via the TCP/IP protocol between the audio peripheral and the PJSIP library on the UE for the generation of the VoIP call's SRTP/SDES stream.

2.2.2 LI Framework Initialization

To configure the OpenLI network as described above, we need to modify the original script *setup.sh* provided by the OpenLI framework to perform the following tasks:

1. Creating the OpenLI networks with IP addresses 172.19.0.0/24 and 172.20.0.0/24, represented in Figure 2.5, and running the simulated containers.

2. Configuring the Collector's interface to receive mobile traffic from the UPF (or gNB, depending on the considered LI implementation).

3. Copying scripts and *yaml* files into the containers to configure Provisioner, Collector, Mediator, and Agency.

Specifically, OpenLI consists of three configuration files, one for each component, and a final file for setting up user interception. In each container, these configuration files are located at the path */etc/openli*.

Firstly, the Provisioner's configuration file, illustrated below (Code 2.5), defines the port for receiving configurations via the REST API (8080), the ports used by the Collector and Mediator to connect to the Provider (9001 and 9002, respectively), and the IP address of the Provisioner.

Code 2.5: provisioner-config.yaml.

```
1 clientport: 9001
2 clientaddr: 172.19.0.3
3 updateport: 8080
4 updateaddr: 172.19.0.3
5 mediationport: 9002
```

```
mediationaddr: 172.19.0.3

intercept-config-file: /etc/openli/running-intercept-config.yaml
```

Secondly, the Mediator's configuration file, detailed in Code 2.6, specifies the IP addresses and ports of Mediator and Provisioner. It also designates the Mediator to receive packets from the Collector on port 12009 and connect to the Provisioner on port 9002 to receive information about the LEA. Additionally, the Operator ID and Mediator ID parameters uniquely identify the network to the Agency and the Mediator within the environment.

Code 2.6: mediator-config.yaml.

```
operatorid: POLIBA

listenport: 12009
listenaddr: 172.19.0.4

provisioneraddr: 172.19.0.3
provisionerport: 9002

mediatorid: 1
```

Thirdly, the Collector's configuration file, depicted in Code 2.7, includes the IP address and port of the Provisioner, along with other parameters such as Operator ID, Network Element ID, Intercept Point ID, thread count for packet encoding and capturing, and the interface for packet interception.

Code 2.7: collector-config.yaml.

```
operatorid: POLIBA
networkelementid: openli-lab
interceptpointid: col001

encoderthreads: 2

inputs:
    - uri: eth2
      threads: 2
```

```
10        hasher: radius
11
12  provisioneraddr: 172.19.0.3
13  provisionerport: 9001
```

Lastly, the file */etc/openli/running-intercept-config.yaml* contains data for running intercepts and regarding LEAs. To modify this file, communication via REST API with the Provisioner is necessary to provide details about the LEA and the user to be intercepted. The commands presented in Code 2.8 are executed on the Provisioner. The first command identifies the LEA by specifying the Agency IP and the ports of the HI2 and HI3 interfaces, while the second command indicates the details of the user to be intercepted. In this case, a Static IP interception is configured in order to monitor user with IP address 10.45.0.3.

Code 2.8: *curl* commands, executed inside the provisioner in order to POST the Agency JSON object and to enable the warrant.

```
1  root@894de214db6f:/home/openli-prov# curl -X POST -H "Content-Type:
       ↪ application/json" -d \
2  > '{
3  >     "agencyid": "Police",
4  >     "hi2address": "172.20.0.3",
5  >     "hi3address": "172.20.0.3",
6  >     "hi2port":"41002",
7  >     "hi3port":"41003",
8  >     "keepalivefreq": 60,
9  >     "keepalivewait": 30
10 > }' http://172.19.0.3:8080/agency
11 <html><body>OpenLI provisioner configuration was successfully
       ↪ updated.</body></html>
12
13 root@894de214db6f:/home/openli-prov# curl -X POST -H "Content-Type:
       ↪ application/json" -d
14 > '{
15 >     "liid": "STATIC002",
16 >     "authcc": "IT",
17 >     "delivcc": "IT",
18 >     "mediator": 1,
19 >     "agencyid": "Police",
```

```
>     "starttime": 0,
>     "endtime": 0,
>     "user": "MarioRossi",
>     "accesstype": "fiber",
>     "staticips": [{"iprange": "10.45.0.4", "sessionid": 101 }]
> }' http://172.19.0.3:8080/ipintercept
<html><body>OpenLI provisioner configuration was successfully
    ↪ updated.</body></html>
```

Moreover, to ensure the operational status and adherence to the desired configuration of the OpenLI containers, one can analyze the log files located in the directory */var/log/openli*. In Codes 2.9, 2.10, and 2.11 there are the logs related to the provisioner, the mediator, and the collector, according to the setup configuration described above.

Code 2.9: OpenLI provisioner logs.

```
OpenLI: provisioner intercepting RTP comfort noise for all VOIP
    ↪ intercepts.
OpenLI: provisioner listening on 172.19.0.3:9001 successfully.
OpenLI: incoming mediator listening on 172.19.0.3:9002 successfully.
OpenLI: update socket listening on 172.19.0.3:8080 successfully.
OpenLI: connection accepted from collector 172.19.0.2-50074
OpenLI provisioner: collector 172.19.0.2-50074 is now active
OpenLI: connection accepted from mediator 172.19.0.4-41288
OpenLI: mediator 172.19.0.4-41288 on fd 11 auth success.
OpenLI: added new agency 'Police' via update socket.
OpenLI provisioner: added new IP intercept STATIC002 via update socket.
```

Code 2.10: OpenLI mediator logs.

```
OpenLI: not using OpenSSL TLS for internal communications.
OpenLI Mediator: '1' has started.
OpenLI: Mediator listening on 172.19.0.4:12009 successfully.
OpenLI Mediator: pcap output file rotation frequency is set to 30 minutes.
OpenLI mediator has connected to provisioner at 172.19.0.3:9002.
OpenLI Mediator: starting pcap output thread.
OpenLI Mediator: received LEA announcement for Police.
OpenLI Mediator: HI2 = 172.20.0.3:41002    HI3 = 172.20.0.3:41003.
OpenLI Mediator: starting agency thread for Police.
OpenLI Mediator: received "Activated" HI1 Notification from provisioner
    ↪ for LIID STATIC002 (target agency = Police).
```

```
OpenLI Mediator: starting collector threads for 172.19.0.2.
OpenLI Mediator: collector threads for 172.19.0.2 have connected to local
    ↪ RMQ instance.
OpenLI Mediator: accepted connection from collector 172.19.0.2.
OpenLI Mediator: LIID STATIC002 has been seen coming from collector
    ↪ 172.19.0.2
OpenLI Mediator: added STATIC002 -> Police to LIID map.
```

Code 2.11: OpenLI collector logs.

```
OpenLI: collector is using a RADIUS-session hasher for input eth2.
OpenLI: collector has started reading packets from eth2 using 2 threads.
openli-collector: all processing threads have reported for duty.
OpenLI: new mediator announcement for 172.19.0.4:12009.
OpenLI: adding new mediator 1 at 172.19.0.4:12009.
OpenLI: received IP intercept for target MarioRossi from provisioner
    ↪ (LIID STATIC002, authCC IT, start time 0, end time 0).
OpenLI: intercepting static IP range 10.45.0.3/32 for LIID STATIC002,
    ↪ AuthCC IT.
```

2.3 IMPLEMENTATION OF THE LI FRAMEWORK IN THE 5G CORE NETWORK

Once the system is running and correctly configured, as extensively discussed in Section 2.2, it is possible to start the interception process within the 5G network. As highlighted in the introduction, the proposed LI framework implemented in this thesis can be flexibly configured to enable the interception of TCP/IP streams (Files Interception) as well as of end-to-end encrypted SRTP streams (VoIP call Interception). The execution phases of these two options are detailed in Sections 2.3.1 and 2.3.2, respectively.

Within the main project folder, the *script* directory contains the bash scripts necessary to conveniently execute one of the two simulation scenarios:

1. *start_all.sh* for setting up the framework.
2. *intercept_all.sh* to initiate interception.
3. *receive_all.sh* and *send_all.sh* for the end-to-end communication.
4. *decipher_all.sh* for the decryption phase of the intercepted stream, which is performed by the LEA.

2.3.1 Files Interception

The specific architecture implemented for file interception is depicted in Figure 2.8, which also succinctly outlines the various phases that occur in the scenario we are about to describe. It is necessary to call the bash files in the script folder with the first argument equal to *"2"*, instructing the framework to reference the bash files contained in the *Files* subdirectory. Let us now consider in detail the various phases implemented by these codes:

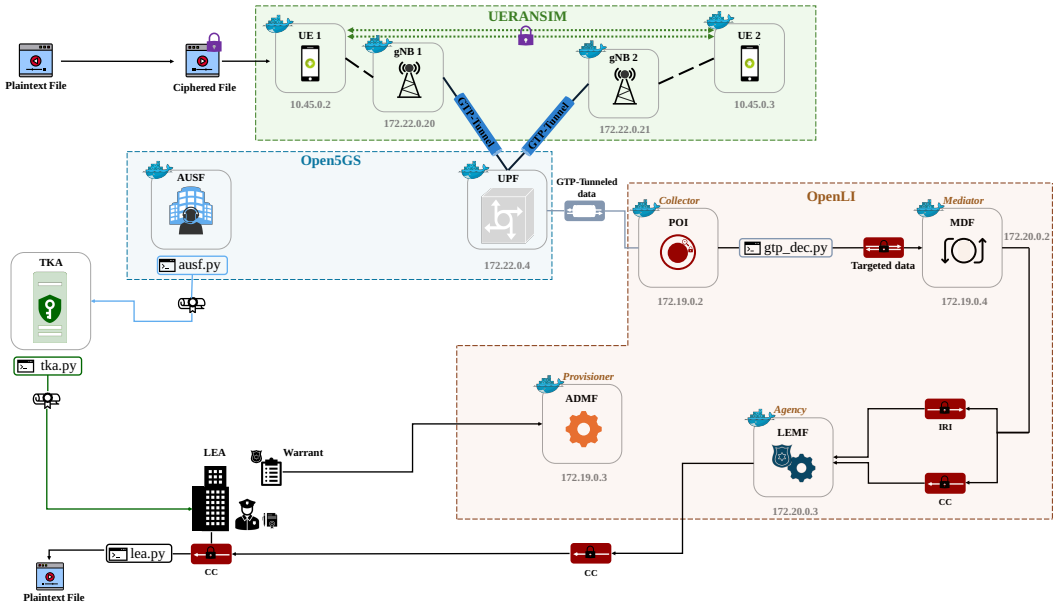
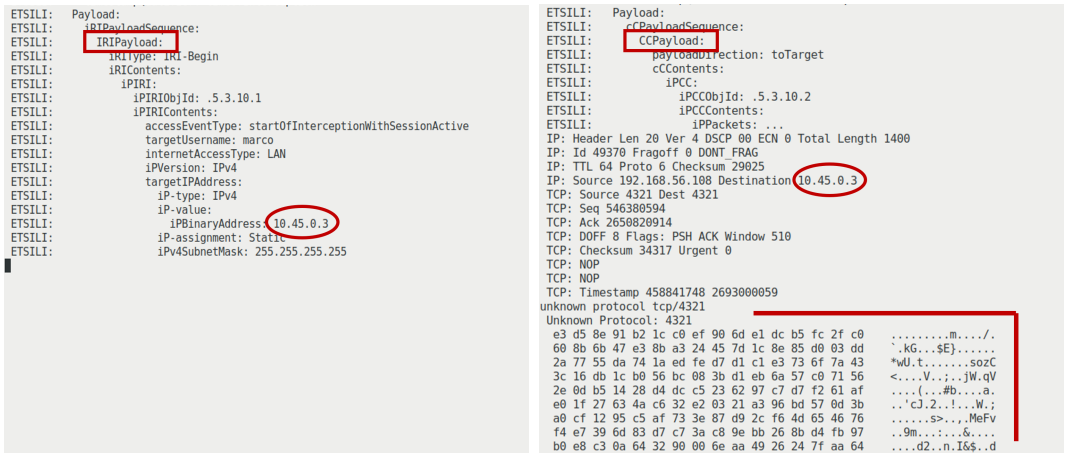


Figure 2.8: Overview of the interception of files in the 5G Core [14].

1. **Interception**: initially, the script *interception_files.sh* is executed to initiate the interception process. The Trusted Key Authority (TKA) generates end-to-end session keys using the IDBC algorithm, which is initialized with two nonces, r_1 and r_2, valid only for the current session and regenerated for each different association. The key material is delivered to UE_1, UE_2, and LEA. After key generation, the LEA, which presented the warrant, is able to receive IRIs and CCs. This is done by executing specific commands (Code 2.12) on two communicated interfaces, HI2 and HI3, to respectively receive IRIs and CCs. Therefore, Figure 2.9 shows an extract of the IRIs and CCs received by the Agency container at the HI2 and HI3 interfaces, respectively, and visualized on the terminal through the *tracepktdump* command.

Code 2.12: Lines of code extracted from *interception_files.sh*.

```
# Receiving and saving interception CCs
sudo docker exec agency tracesplit etsilive:172.20.0.3:41003
    ↪ pcapfile:CC_agency_files.pcap
# Receiving and displaying IRI messages
sudo docker exec agency tracepktdump etsilive:172.20.0.3:41002
```



a b

Figure 2.9: IRIs (a) and CCs (b) captured during the interception and received by the LEMF via the HI2 and HI3 interfaces [14].

2. **End-to-End File Exchage**: firstly, the script *receive_files.sh* (Code 2.13) executes netcat commands on UE_2 to listen on the specified port. Subsequently, the script *send_files.sh* (Code 2.14) is executed on UE_1 to encrypt the file and send it to UE_2. During this phase, it is noteworthy that before being transmitted to the OpenLI Collector, the GTP flow is decapsulated using the Python script *gtp_dec.py*, utilizing the Scapy library. Once the encrypted file is received, UE_2 continues the execution of the *receive_files.sh* Python script to decrypt the received file using the key received from the TKA.

Code 2.13: Lines of code extracted from *receive_files.sh*.

```
local_port=$1
IP_receiver="10.45.0.3"
echo "listening on port $local_port ..."
nc -l -p $local_port > file_received_encr.jpg

# UE_2 keeps waiting for UE_1 to send a file via the open socket

# decryption phase after a file being correctly received
echo "decrypting..."
python3 decrypt_utente.py file_received_encr.jpg
    ↪ file_received_decr.jpg
echo "decryption done!"
```

Code 2.14: Lines of code extracted from *send_files.sh*.

```
remote_port=$1
IP_sender="10.45.0.2"
IP_receiver="10.45.0.3"

# encryption and transmission
echo "encrypting..."
python3 encrypt.py img.jpg file_to_send_encr.jpg
echo "encryption done!"

echo "sending file..."
./nr-binder $IP_sender python3 send_file_user.py send
    ↪ file_to_send_encr.jpg $IP_receiver $remote_port
echo "file sent from $IP_sender to $IP_receiver"
```

3. **Decryption**: the script *decipher_files.sh* is executed to reconstruct the file from the TCP stream received by LEA and properly decrypt it. This script filters and reassembles the packets received by the LEA and then decrypts the intercepted file. Code 2.15 shows agency commands for filtering, reassembling and deciphering the received file, starting from the stream of CCs received from the OpenLI Mediator.

Code 2.15: Lines of code extracted from *decipher_files.sh*.

```
clear_file_path="/home/openli-testagency/deciphered_file.jpg"
# filtering and reassembling
echo "Filtering and reassembling..."
python3 /home/openli-testagency/util/util_files/processing_lea.py /
    ↪ home/openli-testagency/CC_agency_files.pcap

# decryption of the intercepted data
echo "decrypting..."
python3 /home/openli-testagency/util/util_files/lea.py /home/openli-
    ↪ testagency/util/util_files/reassembled_files/reassembled_file
    ↪ .jpg $clear_file_path
echo "done!"
```

2.3.2 VoIP call Interception

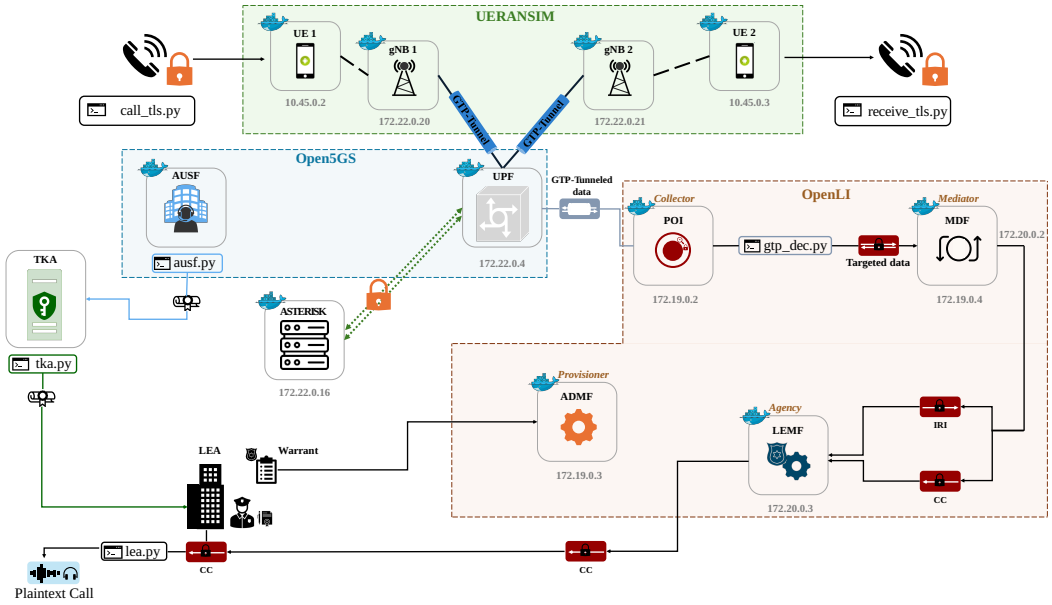


Figure 2.10: Overview of the interception of VoIP calls in the 5G Core [14].

The architecture implemented for VoIP call interception is illustrated in Figure 2.10, summarizing the phases of the forthcoming scenario. To access the relevant scripts, located in the *VoIP* directory, the bash files in the *script* folder should be invoked this time with parameter *"1"*. Let us now go deeper into the detailed phases of the simulation:

1. **Interception**: exactly as it happens with the file interception, the *interception_voip.sh* script initiates the interception by generating end-to-end session keys through the IDBC algorithm. The main difference in this scenario is that these keys get encoded in base64, since this is the format PJSIP requires for the encryption/decryption processes. The keys are valid only for the current session, and they are distributed to UE_1, UE_2, and LEA. Subsequently, commands similar to those in Code 2.12 are executed to allow the LEA to receive IRIs and CCs.

2. **End-to-End VoIP call**: the scripts *receive_call.sh* and *call.sh*, ran in this order, allow UE_1 and UE_2 to correctly establish a SIP session. As evidenced by the bash commands in 2.16, the two scripts utilize the

PJSIP library to make a VoIP call, with UE_1 acting as the caller and UE_2 as the callee. This is simulated by executing the file *call.wav*, running the scripts *call_tls.py* and *receive_tls.py* previously presented in Section 2.1.4. In this stage, it's important to note that prior to transmission to the OpenLI collector, the GTP flow undergoes decapsulation through the Python script *gtp_dec.py*, which utilizes the Scapy library. Once the encrypted file is received, UE_2 executes a Python script to decrypt the received file using the key received from the TKA. The execution of these steps is managed at a high level by PJSUA2 through the invocation of Python functions, differently from what occurred for file deciphering.

Code 2.16: Lines of code extracted from *receive_call.sh* and *call.sh*.

```
# receive the VoIP call
$UE_2_IP=10.45.0.3
sudo docker exec nr_ue_2 ./nr-binder $UE_2_IP python3
    ↪ /UERANSIM/build/receive_tls.py call.wav

# make the VoIP call
$UE_1_IP=10.45.0.2
sudo docker exec nr_ue_1 ./nr-binder $container_ip python3
    ↪ /UERANSIM/build/call_tls.py
```

3. **Decryption**: the LEA, which is able to compute the key used for call encryption, executes the script *decipher_voip.sh* to retrieve the decrypted intercepted call: in turn, this script, runs a C program (*pcaputil.c*) utilizing the PJSIP library, which takes a *pcap* file as input and converts it into a media file. As shown in Code 2.17, to obtain the deciphered audio file, the encryption algorithm and key need to be specified in order to filter, reassemble, and decrypt the intercepted SRTP stream using the escrowed key provided by the TKA.

Code 2.17: Lines of code extracted from *decipher_voip.sh*.

```
# key received from TKA
key=$(cat /home/openli-testagency/keyLEA.txt)
```

```
3
4  # command to retrieve the ciphered VoIP call
5  ./pcaputil --src-ip=10.45.0.2 --src-port=4000
       ↪ /home/openli-testagency/CC_agency_enc.pcap
       ↪ /home/openli-testagency/CC_cipher.wav
6
7  # command to decrypt the SRTP VoIP call
8  ./pcaputil --src-ip=10.45.0.2 --src-port=4000 -c
       ↪ AES_CM_128_HMAC_SHA1_80 -k $key
       ↪ /home/openli-testagency/CC_agency_enc.pcap
       ↪ /home/openli-testagency/CC_decipher.wav
```

2.4 IMPLEMENTATION OF THE LI FRAMEWORK AT THE EDGE OF THE 5G NETWORK

Multi-access Edge Computing (MEC) technology, as pointed out in Section 1.2.3, is one of the key enablers of the 5G technology. It facilitates the targeted placement of content, applications, and services by relocating computing storage and business service capabilities at the network edge. This enhances the efficiency of task execution and ensures compliance with users' stringent delay requirements through the deployment of high-performance MEC servers beside the RAN [4].

In the realm of Lawful Interception, this concept could be integrated to enhance the architecture outlined in the preceding section. Indeed, the idea pursued by the proposed LI framework is to develop an interception framework that involves the activation of POIs located within the 5G NR, particularly at the gNB level. Therefore, the decapsulation and forwarding of IRIs and CCs would occur at the network edge, lightening the workload of the UPF, while also reducing the latency perceived by the LEA.

2.4.1 Implementation of the cutting-edge LI framework

In order to perform Lawful Interception within the 5G NR network segment, the following modifications to the project are implemented:

1. Creation of a new network that establishes communication between gNB_2 and the POI, designated as the *openli-poi* network:

```
# from the .env file
OPENLI_NETWORK_EDGE=172.23.0.0/24
COLLECTOR_IP_EDGE=172.23.0.2

# creating the edge network
sudo docker network inspect openli-poi > /dev/null 2>&1 || \
        sudo docker network create --driver bridge \
        -o "com.docker .network.bridge.enable_icc=true" \
        --subnet $OPENLI_NETWORK_EDGE \
        openli-poi

# attaching the OpenLI collector to the openli-poi network
sudo docker network connect --ip $COLLECTOR_IP_EDGE openli-poi
    ↪ collector
```

This new network replaces the connection between the *ogstun* interface of the UPF and the *eth2* interface of the POI. Accordingly, the new interfaces on the gNB_2 and the OpenLI Collector need to be reconfigured in their respective configuration files *nr-gnb.yaml* and *collector-config.yaml*, as shown in Codes 2.18 and 2.19.

Code 2.18: The updated version of*nr-gnb.yaml*.

```
services:
  nr_gnb1:
    [...]
  nr_gnb2:
    image: docker_ueransim
    container_name: nr_gnb_2
    stdin_open: true
    tty: true
    volumes:
```

```
        - ./ueransim:/mnt/ueransim
        - /etc/timezone:/etc/timezone:ro
        - /etc/localtime:/etc/localtime:ro
    env_file:
        - .env
    environment:
        - COMPONENT_NAME=ueransim-gnb-2
    expose:
        - "38412/sctp"
        - "2152/udp"
        - "4997/udp"
    cap_add:
        - NET_ADMIN
    privileged: true
    networks:
        default:
            ipv4_address: ${NR_GNB_IP_2} # 172.22.0.21
        li:
            ipv4_address: ${NR_GNB_IP_OPENLI} # 172.23.0.3
    networks:
      default:
        name: open5gs_default
        external: true
    li:
      name: openli-poi
      external: true
```

Code 2.19: The updated version of *collector-config.yaml*.

```
operatorid: POLIBA
networkelementid: openli-lab
interceptpointid: col001

encoderthreads: 2

inputs:
    - uri: eth3   # new interface with the gNB_2
    threads: 2
    hasher: radius

provisioneraddr: 172.19.0.3
provisionerport: 9001
```

2. Reconfiguration of the *iptables* rules for network traffic mirroring towards the POI, a task now performed by the gNB instead of the UPF. Code 2.20 shows the new IP routing rules for mirroring the traffic passing through the gNB_2 (in both directions) towards the *openli-poi* network.

Code 2.20: IP routing rules added for acheving LI at gNB_2.

```
1 COLLECTOR_IP_EDGE=172.23.0.2
2 sudo docker exec nr_gnb_2 iptables -t mangle -A PREROUTING -i eth0
      ↪ -j TEE --gateway $COLLECTOR_IP_EDGE
3 sudo docker exec nr_gnb_2 iptables -t mangle -A POSTROUTING -o eth0
      ↪ -j TEE --gateway $COLLECTOR_IP_EDGE
```

2.4.2 Files Interception

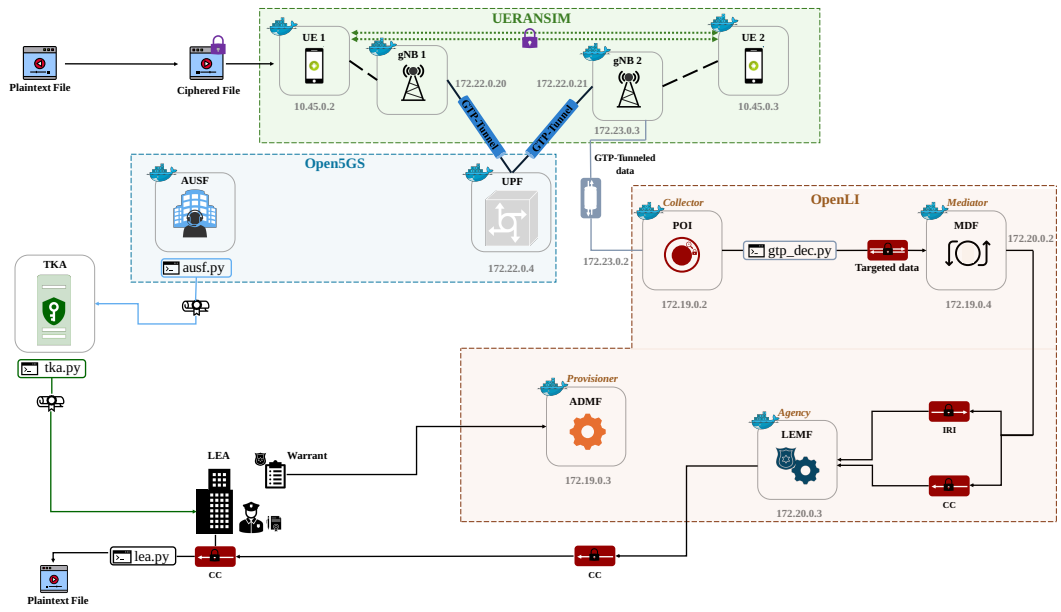


Figure 2.11: Overview of the interception of files in the 5G NR.

In addition to providing a concise overview of the simulation steps, Figure 2.11 depicts the specific architecture adopted for the interception of files. Here, the proposed LI framework is directed to reference the bash files in

the *Files* subdirectory by invoking the scripts with the first argument equal to "2". Let us now detail the phases of the scenario:

1. **Interception**: the interception process starts with the execution of the script *interception_files.sh*. The TKA utilizes the IDBC algorithm to generate end-to-end session keys. Two nonces, r_1 and r_2, are initialized and remain valid only for the ongoing session; fresh nonces are generated for each new session. Then, the LEA, UE_1, and UE_2 obtain the key material. The LEA, upon presenting the warrant, is allowed to receive IRIs and CCs subsequent to key generation, thus executing specific commands to deploy the HI2 and HI3 communication interfaces.

2. **End-to-End File Exchange**: the script *receive_files.sh* instructs UE_2 to listen on the designated port using netcat commands. Subsequently, the script *send_files.sh* is executed on UE_1 to encrypt the file and transmit it to UE_2. It is noteworthy that, at this stage, the Python script *gtp_dec.py* uses the Scapy library to decapsulate the GTP flow before forwarding it to the OpenLI Collector. Upon receiving the encrypted file, UE_2 runs a Python script to decrypt the file using its session key.

3. **Decryption**: to effectively decrypt and reconstruct the file from the TCP stream received by LEA, the script *decipher_files.sh* is executed. This script decrypts the intercepted file after filtering and reassembling the received packets.

2.4.3 VoIP call Interception

The architecture for VoIP call interception is outlined in Figure 2.12, which also delineates the stages of the simulation. The bash files in the *script* folder need to be run with "1" as first parameter to access the related

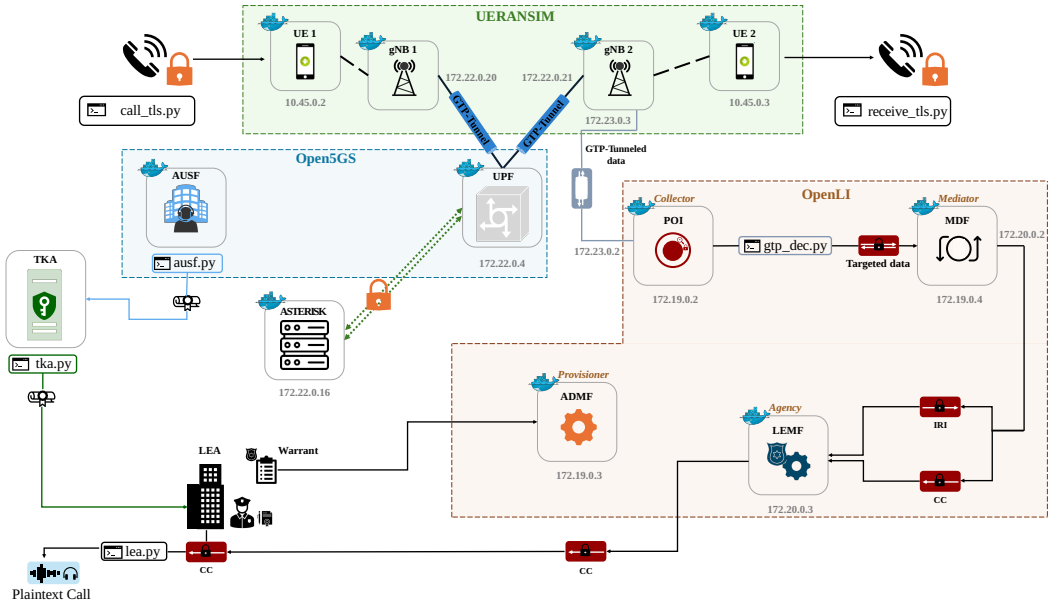


Figure 2.12: Overview of the interception of VoIP calls in the 5G NR.

scripts located in the *VoIP* directory. The main steps of the process could be schematically outlined as follows:

1. **Interception**: the *interception_voip.sh* script initiates interception by utilizing the IDBC algorithm to generate end-to-end session keys, similar to the file interception process. Notably, in this instance, as required by PJSIP for encryption and decryption operations, these keys are encoded in base64 format. Distributed to UE_1, UE_2, and LEA, these keys remain valid only for the ongoing session. Subsequently, commands similar to those in Code 2.12 are executed to enable the LEA to receive IRIs and CCs.

2. **End-to-End VoIP Call**: the scripts *call.sh* and *receive_call.sh* are executed in sequence to allow UE_1 and UE_2 to establish a SIP session successfully. These scripts use the PJSIP library to simulate a VoIP call, as evidenced by the bash commands in Code 2.16. They execute the file *call.wav* and run the scripts *call_tls.py* and *receive_tls.py*, previously introduced in Section 2.1.4. At this point, the Python script *gtp_dec.py* utilizes the Scapy library to decapsulate the GTP flow before transmission to the OpenLI Collector. Upon receiving the encrypted file,

UE_2 utilizes the key obtained from TKA to execute a Python script for decoding the ciphertext. Unlike file processing, which operated at a lower level, PJSUA2 manages the execution of these phases at a higher level through Python functions.

3. **Decryption**: the base64-encoded key used for call encryption, computed by the LEA, is used to decrypt the intercepted call upon running the script *decipher_voip.sh*. As demonstrated in Code 2.17, the encryption algorithm and the key are mandatory to listen the audio file in clear.

3 PERFORMANCE EVALUATION

This chapter deals with the performance analysis of the proposed Lawful Interception framework developed, whose theoretical and implementation details have been provided in the previous chapters. In particular, the following section analyses the significant potential of the proposed LI framework and its implementation through experimental testing [14]. The testbed is set up on a workstation with a processor Intel® Core™ i5-9400 CPU @ 2.90 GHz and 16 GB of RAM. The impact of the number of packets processed and the stream duration on the latency involved in the lawful interception procedure is being evaluated. This includes assessing the delay perceived by the two communicating User Equipments (UEs) in scenarios where the LI architecture is deployed or not, examining the Quality of Service (QoS). In this regard, five Key Performance Indicators (KPIs) are considered [14]:

1. **UPF acquisition latency**: the time required for each packet to be processed by the UPF, serving as a reference of the 5GC workload, since the UPF is not part of the nodes involved in the interception.

2. **POI capturing latency**: the time needed for each packet to be intercepted by the Collector.

3. **LEMF collecting latency**: it measures the time taken for each targeted packet to be delivered to the Agency.

4. **End-to-end LI Latency**: it represents the time required for each packet to be processed by the system during the overall interception mecha-

nism, calculated as the sum of the POI capturing and LEMF collecting latencies.

5. **End-to-end User Latency**: this metric defines the end-to-end packet-per-packet delay experienced by UE_2 in receiving the stream from UE_1, or vice versa.

The following sections are organized to present the analysis of the data collected in the two scenarios where the lawful interception procedures are performed in the 5GC or in the 5G NR portion of the network. For each of these two scenarios, the network performances are evaluated when either a file exchange or VoIP call occurs between the two UEs.

3.1 PERFORMANCES OF THE PROPOSED LI FRAMEWORK IN THE 5G CORE

The assessment involves the usage of the aforementioned simulation configuration to evaluate file exchanges of four different increasing sizes (i.e., 10 KB, 10^2 KB, 10^3 KB, and 10^4 KB) and also VoIP calls of varying time durations (15, 30, 45, and 60 seconds). Since each of the four file dimensions and call durations undergoes 10^2 simulations over multiple seeds, the resulting average values are used as KPIs [14].

3.1.1 Analysis of the End-to-end LI Latency per packet

To further accelerate the performance analysis of the proposed LI framework, several *pcap* files are arranged to be captured within the network architecture, each with the purpose of keeping track of the processing timestamps of each packet within the 5G and LI architecture. In detail, for the latency analysis of the lawful interception operations, traffic is captured at the POI's and LEA's appropriate interfaces. The global amount of data collected is then processed with the aid of Python and Matlab scripts as to extrapolate and graphically arrange the considered KPIs.

Real-time file exchange

As presented in [14], Figures 3.1 and 3.2 show the average latency for each packet across the LI phases when a 10^3 KB file is exchanged between the two UEs. Since both POI capturing and LEMF collecting require a per-packet processing time steadily between 0.5 ms and 0.25 ms, it is possible to consider that neither phase decisively affects the End-to-end LI Latency compared to the other, as the differences in latency of the two phases are negligible.

As a result, Figure 3.3 shows that the End-to-end LI Latency for each packet, computed as the sum of POI capturing and LEMF collecting graphs, consistently fluctuates around 0.20 ms, with almost every packet experiencing latency times below 0.45 ms. However, it is noteworthy that most of the targeted packets are conveyed to the LEA in less than 0.45 ms, thus highlighting how the proposed framework is able to handle real-time interceptions during end-to-end file transfers.

To complete the latency analysis and extend the considerations to the other transmitted files, Figure 3.4 shows the statistics of how the size of the transmitted file impacts, packet by packet, the End-to-end LI Latency. The graph jointly shows the mean value of the End-to-end LI Latency and the box plot of its statistical distribution, highlighting 25^{th}, 50^{th}, and 75^{th} percentiles. It is worthy of mention that the file size has a significant impact on the minimum and maximum latencies that each packet experiences, since it is observed that the delay varies between 0.07 ms and 0.10 ms for small files (10 KB), and between 0.08 ms and 0.4 ms for larger files.

In fact, Figure 3.4 also demonstrates that the file size determines the average end-to-end LI delay per packet, which rises with the size of the transferred file. Despite the average delay doubling from a 10 KB file to a 10^4 KB file, even with a 10^4 KB file the average packet end-to-end LI latency at the LEA side is slightly above 0.25 ms, thus guaranteeing that the LEA processes, on average, each packet in real time.

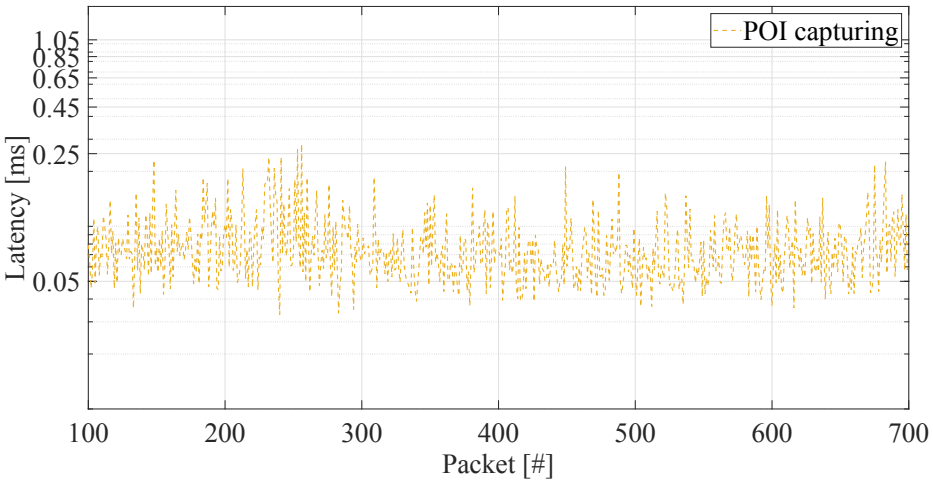


Figure 3.1: Packet-wise POI capturing latency for a 10^3 KB exchanged file [14].

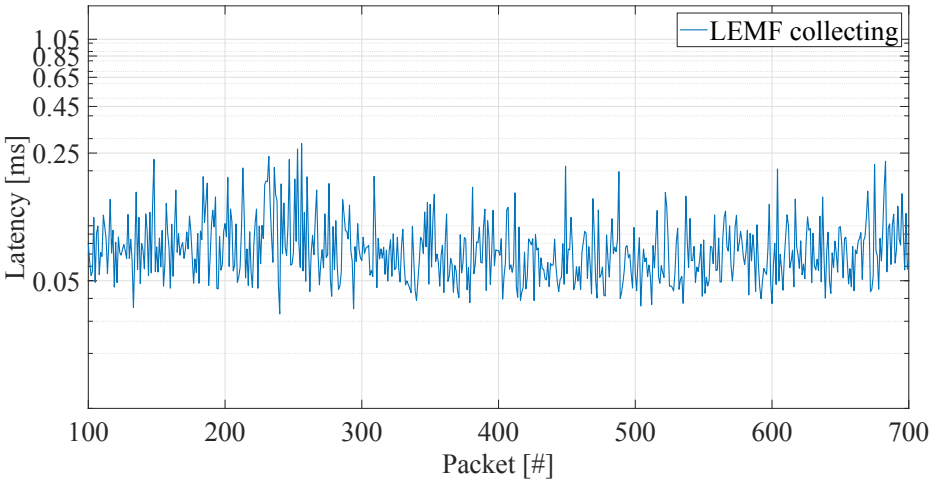


Figure 3.2: Packet-wise LEMF collecting latency for a 10^3 KB exchanged file [14].

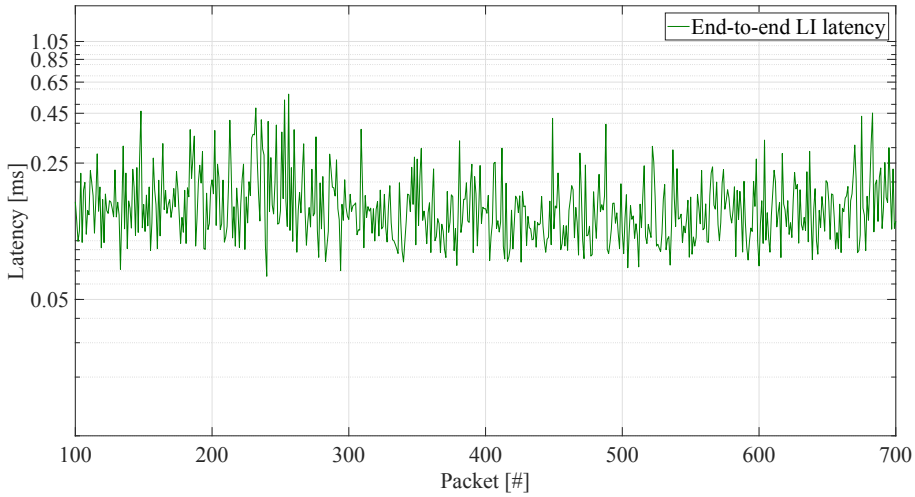


Figure 3.3: Packet-wise End-to-end LI latency for a 10^3 KB exchanged file [14].

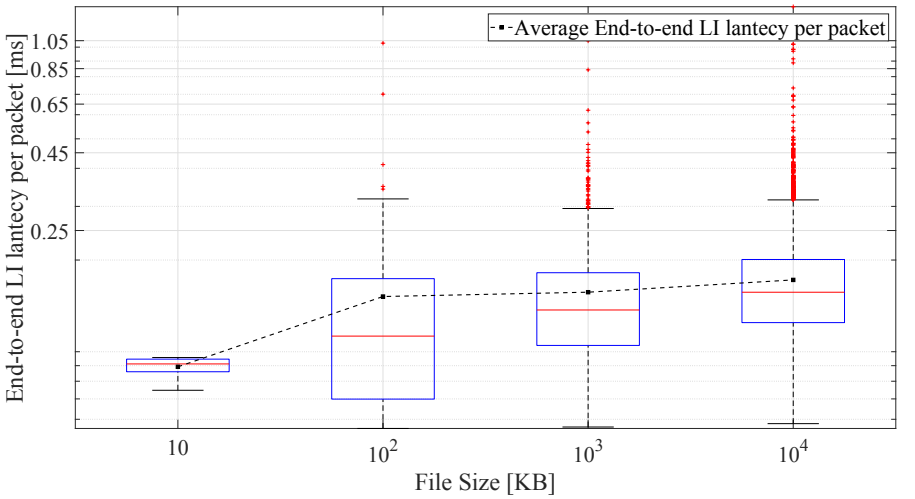


Figure 3.4: Statistics of the End-to-end LI latency per packet as a function of the four file sizes [14].

Real-time VoIP call

Concerning the study of the performances of a VoIP call interception within the 5GC [14], Figures 3.5 and 3.6 display the average latency for each SRTP packet within a 30-second VoIP call when reaching the POI and the LEMF, respectively. It is evident that POI capturing and LEMF collecting latencies are essentially constant at 20 ms, and only a minimal amount of packets has latency exceeding 30 ms. Furthermore, analyzing the End-to-end LI Latency (Figure 3.7), the majority of the SRTP packets are transmitted to the LEA in less than 0.07 s, proving that the system is able to manage real-time interceptions also during VoIP calls.

Secondly, for the sake of clarity, Figure 3.8 extends the study to the different VoIP call durations simulated, evaluating the impact of the VoIP call duration on the entire interception process. Indeed, the graph shows information about the distribution of the End-to-end LI Latency per packet and its average, by highlighting the 25^{th}, 50^{th}, and 75^{th} percentiles, as well as the minimum and maximum values of the End-to-end LI Latency reached by each packet. As a result, Figure 3.8 proves that there is no significant performance degradation when the VoIP call has a longer duration, because the End-to-end LI Latency for each packet typically falls between 35 ms and 60 ms. Additionally, Figure 3.8 indicates that the average End-to-end LI Latency per packet remains consistently close to 40 ms throughout the four VoIP calls, thereby affirming the scalability of the proposed methodology.

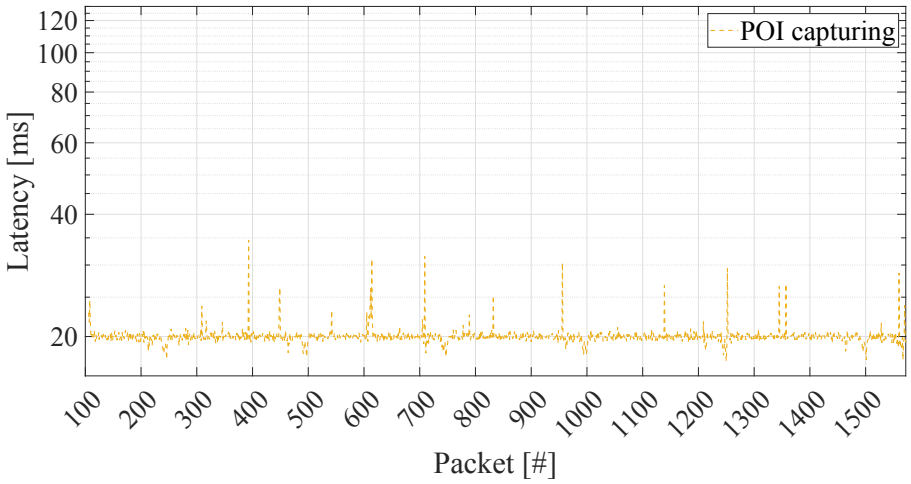


Figure 3.5: Packet-wise POI capturing latency for a 30-second VoIP call [14].

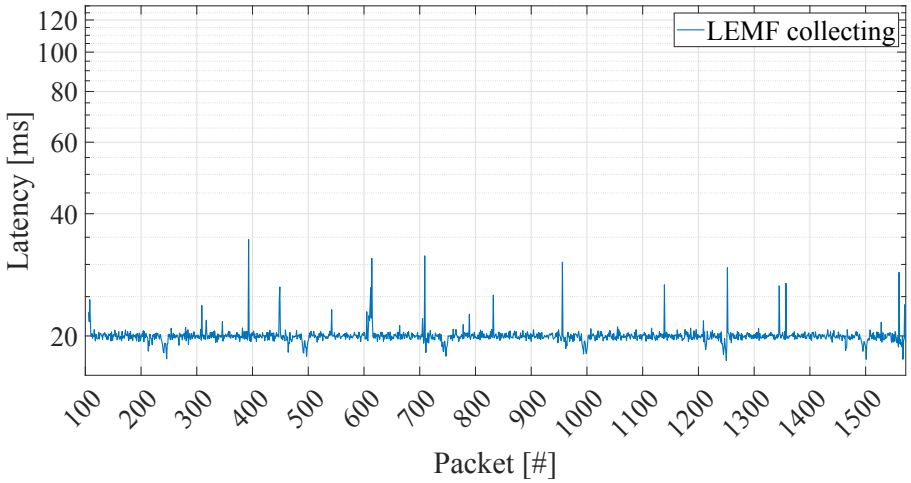


Figure 3.6: Packet-wise LEMF collecting latency for a 30-second VoIP call [14].

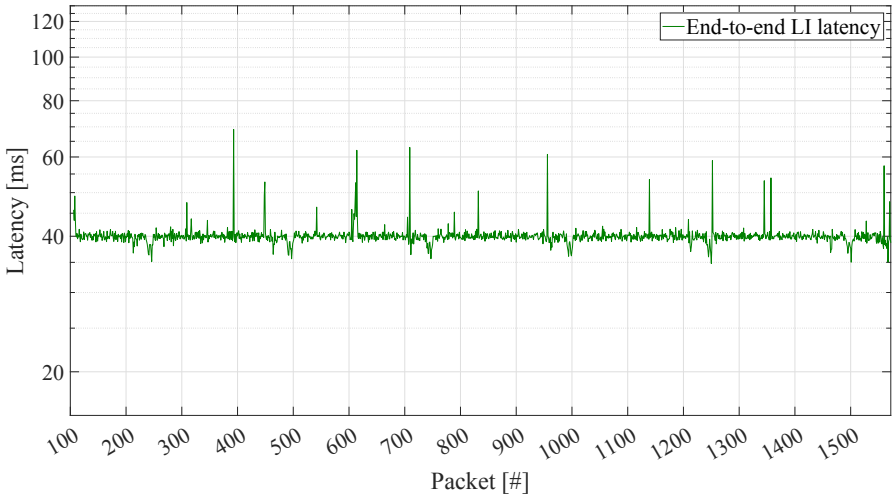


Figure 3.7: Packet-wise End-to-end LI latency for a 30-second VoIP call [14].

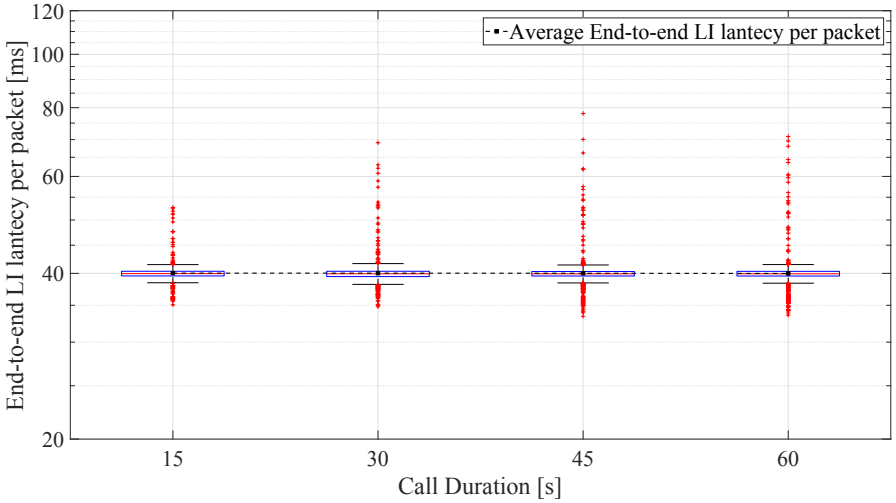


Figure 3.8: Statistics of the End-to-end LI latency phase per packet for the four different VoIP call durations [14].

3.1.2 Impact of LI on the user Quality of Service (QoS)

For the evaluation of the QoS perceived by the two User Equipments when the proposed LI framework is or is not deployed, the *pcap* files obtained during the simulations at the interfaces of the two User Equipments' containers are analyzed [14].

Figure 3.9 describes the packet end-to-end user latency KPI for a 10^3 KB exchanged file. The graph shows that in the case the deployment of the proposed LI framework, there is a slight increase in the delay perceived by the user compared to the absence of an active LI framework. Specifically, Table 3.1, which provides more detail on the average delay difference experienced by each packet, shows that the mean increment perceived per each packet is approximately 31 nanoseconds. This data allow us to state the perfect compatibility of the proposed LI framework within real-time interception scenarios, since there is no significant and recognizable change in the QoS. Moreover, the detected, yet negligible, difference between the two curves in Figure 3.9 can be attributed to the fact that, although the proof-of-concept uses standalone containers, it runs on a single workstation.

Regarding the VoIP call, Figure 3.10 illustrates the end-to-end user latency experienced by each packet received by UE_2 during a 30-second VoIP call, highlighting that in the proposed LI framework presented in this thesis project there is no significant impact on the delay perceived by the user. In fact, for a significant portion of the simulation, the average delay experienced by UE_2 in receiving each packet is essentially the same regardless of whether the LI framework is active or inactive. As presented in Table 3.1, the average delay experienced by each packet is approximately 38 microseconds.

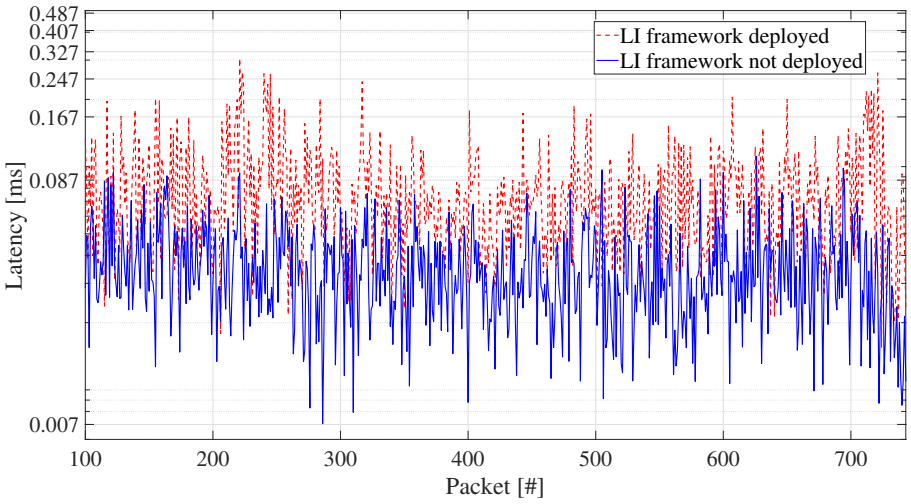


Figure 3.9: Packet-wise End-to-End latency of a 10^3 KB exchanged file [14].

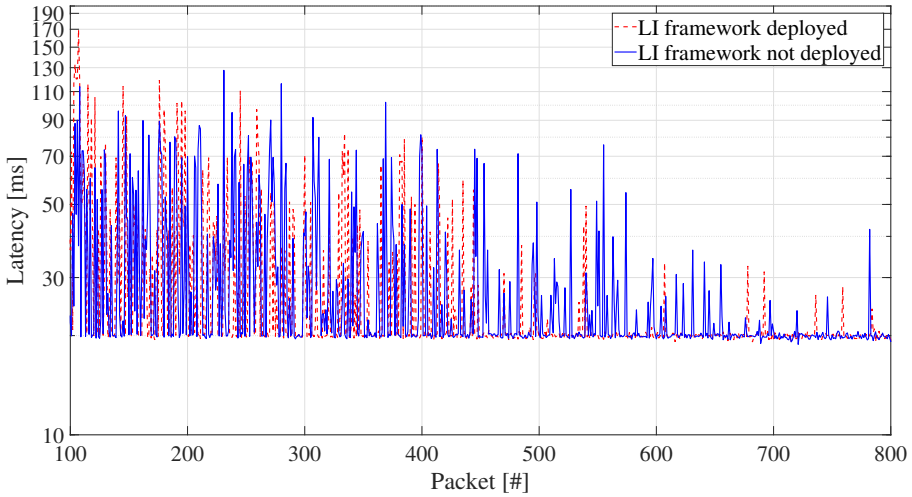


Figure 3.10: Packet-wise End-to-End latency of a 30-second VoIP call [14].

Table 3.1: Average delay difference experienced by each packet when the LI framework is deployed or not in the 5G Core [14].

FILE EXCHANGE

File Size [KB]	Average delay difference experienced by a single packet [ms]
10	0.000012
10^2	0.000017
10^3	0.000031
10^4	0.000035

VoIP CALL

Call Duration [s]	Average delay difference experienced by a single packet [ms]
15	0.077452
30	0.038487
45	0.033085
60	0.049420

3.2 PERFORMANCES OF THE PROPOSED LI FRAMEWORK IN THE 5G NEW RADIO

In order to assess the performance of the proposed cutting-edge LI framework where the POI is situated at the network edge and to obtain metrics comparable to those outlined in Section 3.1, we conducted simulations using the same file types (with dimensions of 10 KB, 10^2 KB, 10^3 KB, and 10^4 KB) and VoIP calls (with durations of 15, 30, 45, and 60 seconds). After running 10^2 simulations for each file and VoIP call category, the resulting KPIs are computed and averaged.

3.2.1 Analysis of the End-to-end LI Latency per packet

In the study of packet-wise end-to-end LI latency when the LI framework is deployed at the edge of the 5G network, similarly to the 5GC scenario, *pcap* files are generated within the 5G network to track information about each packet. Specifically, for the latency analysis of lawful interception procedures, traffic is captured at the POI's and LEMF's appropriate interfaces. The gathered data is processed using Python and Matlab scripts to compute and visually display the KPIs of interest.

Real-time file exchange

Figures 3.11 and 3.12 show the average latency for each packet across the two LI phases considered when a 10^3 KB file is exchanged between the two UEs. Given that the processing times for POI capturing and LEMF collecting consistently hover around 0.25 ms, it can be reasonably concluded that neither phase significantly impacts the End-to-end LI Latency due to the minimal differences in latency between the two phases.

Furthermore, Figure 3.13 shows the end-to-end LI Latency for each packet, which is the sum of the times required for POI capturing and LEMF collect-

ing. The graph demonstrates that the end-to-end LI Latency consistently falls between 0.45 ms and 0.65 ms, with only a few outliers exceeding this range. Nonetheless, the majority of targeted packets are delivered to the LEA in less than 0.65 ms, suggesting the system's ability to handle real-time interceptions during end-to-end file transfers.

To further extend the analysis, Figure 3.14 shows the relationship between the size of the transmitted file and the corresponding End-to-end LI Latency. The figure jointly illustrates the mean values and the box plots of the End-to-end LI Latency distribution, in particular the 25^{th}, 50^{th}, and 75^{th} percentiles. As each packet experiences a minimum and maximum latency determined by the size of the corresponding file, it is important to note that the experienced delay oscillates between 0.32 ms and 0.48 ms for small files (10 KB) and between 0.3 ms and 0.8 ms for the bigger ones.

Moreover, Figure 3.14 outlines how the average end-to-end LI delay per packet increases in proportion to the size of the exchanged file, underscoring the dependence from the file's dimension. Anyway, even if the average delay almost doubles when increasing the size of the file from 10 KB to 10^2 KB, the average packet end-to-end LI latency at the LEA side remains steady slightly below 0.65 ms, thus guaranteeing that the LEA averagely processes each packet in real time.

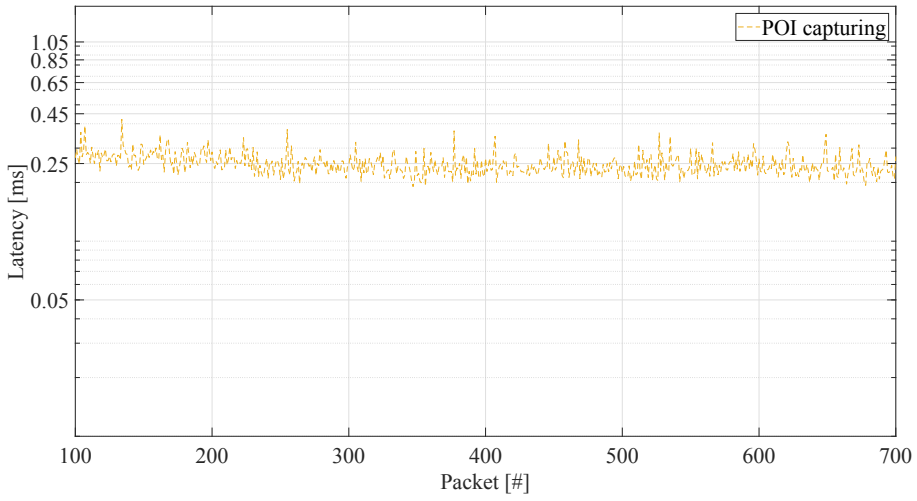


Figure 3.11: Packet-wise POI capturing latency for a 10^3 KB exchanged file.

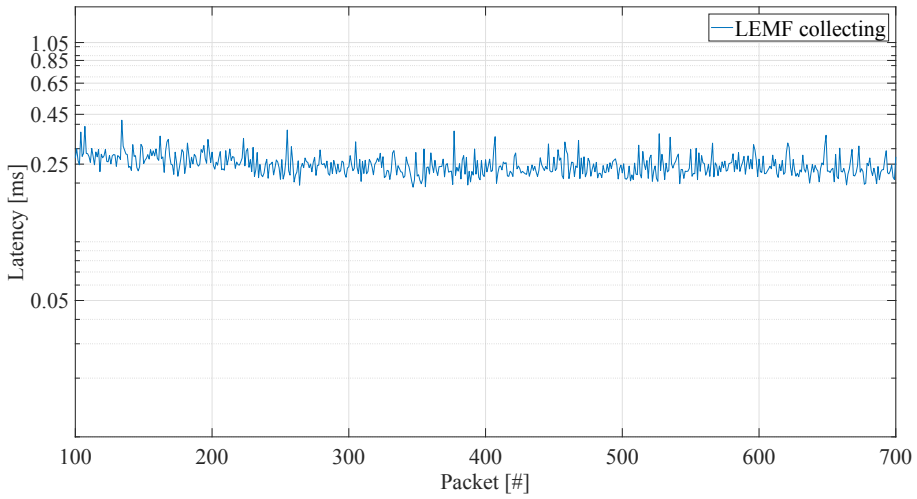


Figure 3.12: Packet-wise LEMF collecting latency for a 10^3 KB exchanged file.

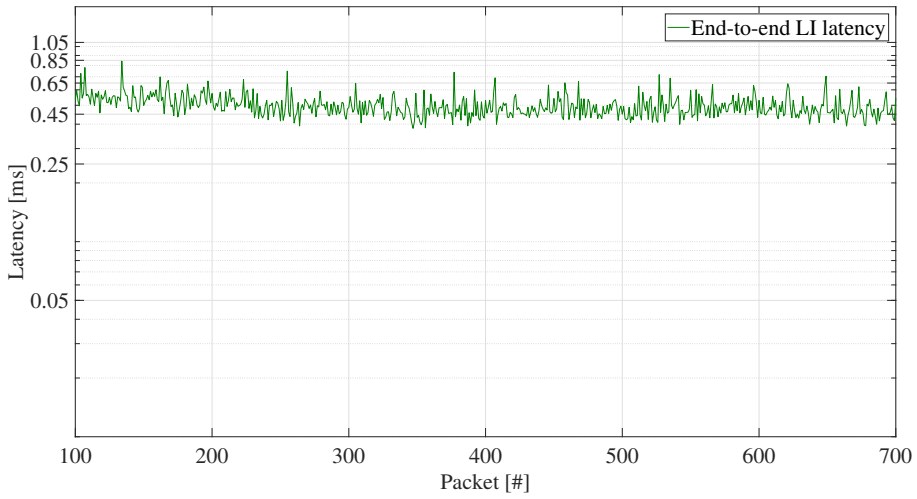


Figure 3.13: Packet-wise End-to-end LI latency for a 10^3 KB exchanged file.

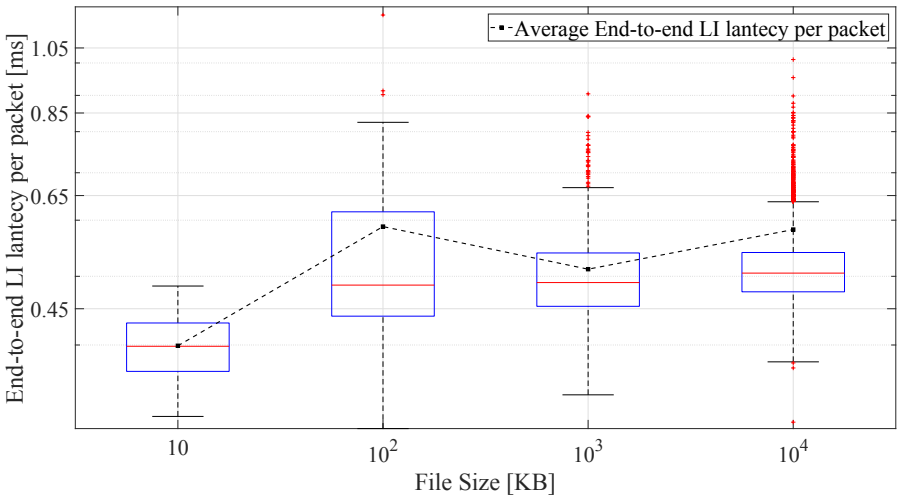


Figure 3.14: Statistics of the End-to-end LI latency phase per packet as a function of the four file sizes.

Real-time VoIP call

In the real-time VoIP call scenario, the average latency for each SRTP packet in a 30-second VoIP call is illustrated in Figures 3.15 and 3.16, by describing the packet-by-packet performance of a VoIP call interception within the 5G NR.

The two graphs show that POI capturing and LEMF collecting latencies are essentially within the interval 5-20 ms, both characterized by marked fluctuations and rare spikes above 40 ms. In addition, analyzing the combination of the described two metrics, that is the End-to-end LI Latency (Figure 3.17), a large portion of the SRTP packets are transmitted to the LEA in less than 40 ms, demonstrating that the system allows real-time interceptions of VoIP communications.

Moreover, Figure 3.18 describes how the different VoIP call durations influence the per-packet overall End-to-end LI Latency. Indeed, the graph conveys information about both the distribution of the End-to-end LI Latency per packet and its average, by emphasizing the 25^{th}, 50^{th}, and 75^{th} percentiles, as well as its highest and lowest values. Consequently, the box plots in Figure 3.18 demonstrates that longer VoIP calls often do not re-

sult in higher delays, since the End-to-end LI Latency distribution remains quite unaffected and with values ranging from 2 to 10 ms. Furthermore, Figure 3.18 demonstrate that the average End-to-end LI Latency per packet consistently remains steady to 6 ms during any of the VoIP calls, verifying the scalability of the proposed methodology.

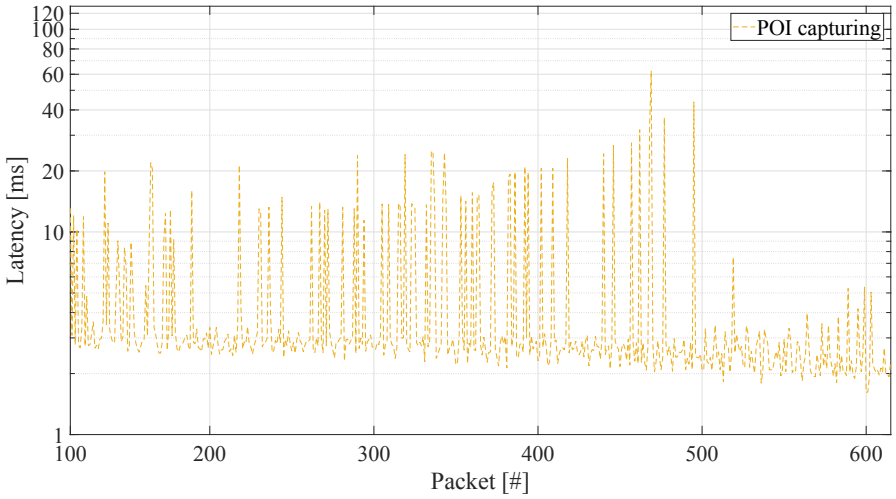


Figure 3.15: Packet-wise POI capturing latency for a 30-second VoIP call.

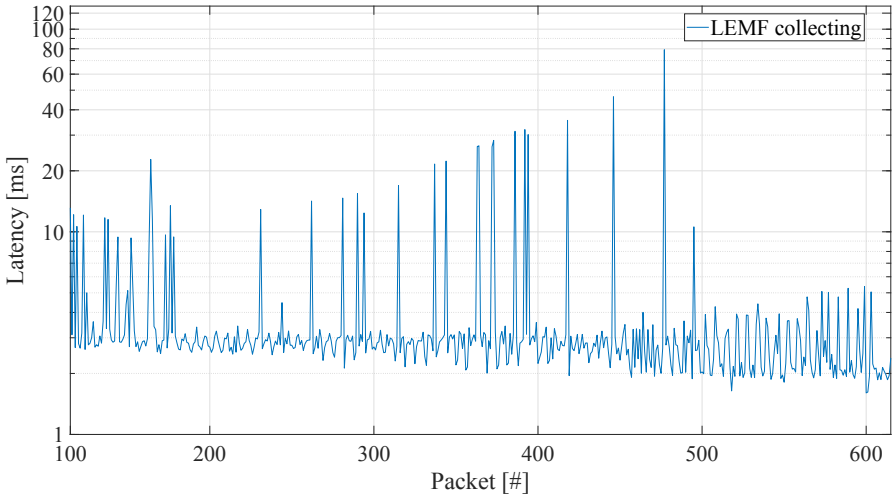


Figure 3.16: Packet-wise LEMF collecting latency for a 30-second VoIP call.

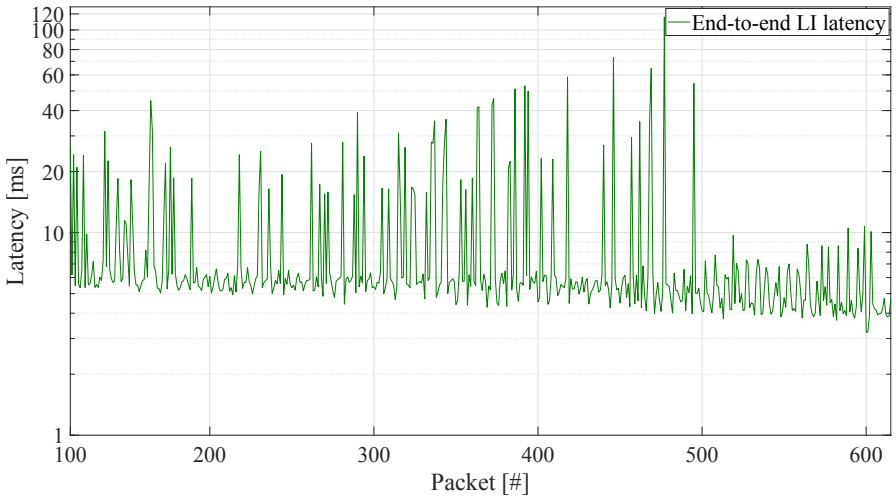


Figure 3.17: Packet-wise End-to-end LI latency for a 30-second VoIP call.

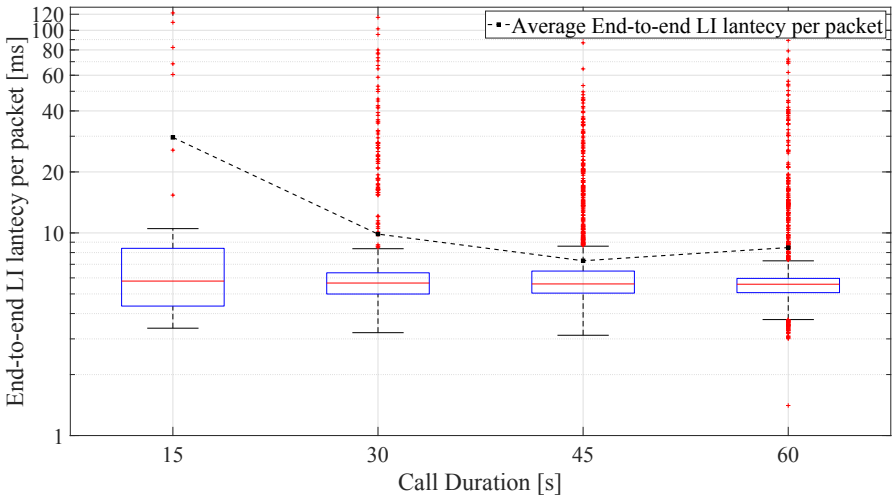


Figure 3.18: Statistics of the End-to-end LI latency per packet for the four different VoIP call durations.

3.2.2 Impact of LI on the user Quality of Service (QoS)

As for the study of LI in the 5GC (Section 3.1.2), to analyze the KPI related to the QoS experienced by the two User Equipments when the LI framework is or is not installed in the 5G NR, we evaluate the *pcap* files collected during the simulations at the interfaces of the two User Equipments' containers.

At first, the packet end-to-end user latency KPI for a 10^3 KB file is shown in Figure 3.19. The graph indicates that, when the proposed Lawful Interception framework is implemented at the network edge, the user experiences a slightly higher latency than when the LI framework is inactive. The mean value of this increment per packet is about 35 nanoseconds, as shown in Table 3.2 which gives further information on the average delay difference experienced by each packet with and without the LI framework. Since the QoS is not significantly affected, we can state that the proposed LI framework is compatible with real-time interception scenarios. The reason behind this small variation between the two curves in Figure 3.19 is that the entire proof-of-concept was implemented on a single workstation.

Secondly, Figure 3.20 illustrates the end-to-end user latency experienced by each packet received by UE_2 during a 45-second VoIP call. This plot illustrates that the proposed LI framework does not significantly affect the user's perceived delay. In fact, regardless of whether the LI framework is active or not, UE_2 reports approximately the same average delay in receiving each packet, namely 38 microseconds (Table 3.2).

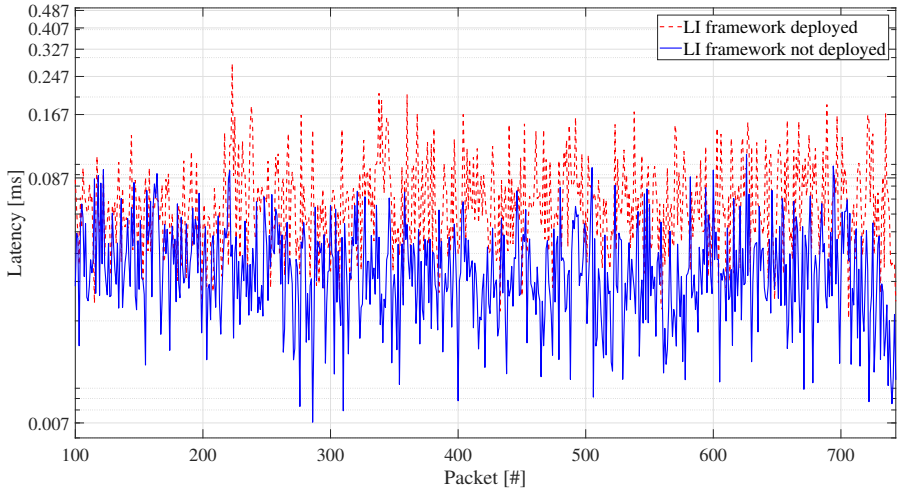


Figure 3.19: Packet-wise End-to-End latency of a 10^3 KB exchanged file.

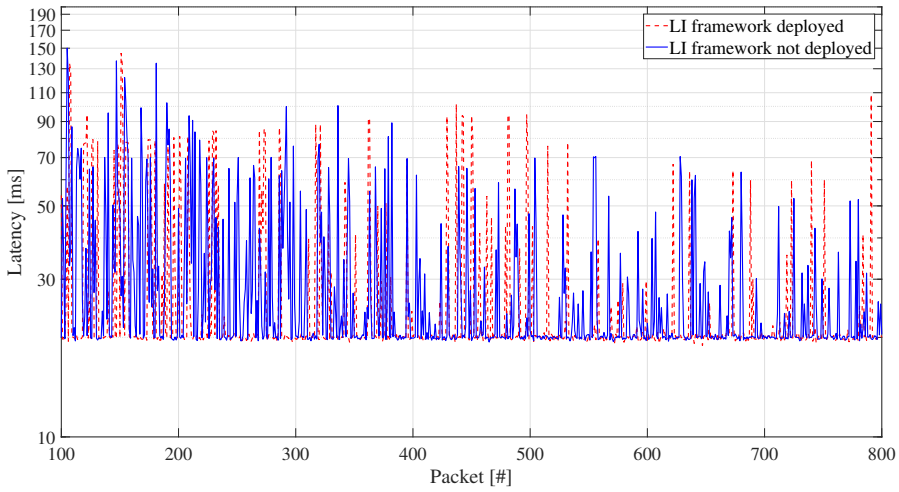


Figure 3.20: Packet-wise End-to-End latency of a 45-second VoIP call.

Table 3.2: Average delay difference experienced by each packet when the LI framework is deployed or not in the 5G NR.

FILE EXCHANGE

File Size [KB]	Average delay difference experienced by a single packet [ms]
10	0.000025
10^2	0.000032
10^3	0.000035
10^4	0.000035

VoIP CALL

Call Duration [s]	Average delay difference experienced by a single packet [ms]
15	0.041401
30	0.033066
45	0.025078
60	0.027489

3.3 COMPARATIVE ANALYSIS AND FINAL CONSIDERATIONS

After analyzing and describing the KPIs considered in the performance evaluation of the two implementations of the proposed LI framework, this section compares the obtained results previously presented in Sections 3.1 and 3.2. The aim is to formulate overall evaluations regarding the advantages of moving the workload of LI procedures to the 5G network edge, but also to underline the limitations of the study carried out in order to arrive at objective final considerations.

Table 3.3: Summary of UPF processing times and overall LI Latency per packet for a 10^3 KB files and a 30-second VoIP call, which compares performances of the proposed LI framework in the 5G Core and in the 5G NR.

	Average UPF processing time (per packet)		**Average End-to-end LI latency (per packet)**	
	LI in the 5G CN	***LI in the 5G NR***	***LI in the 5G CN***	***LI in the 5G NR***
File Exchange	0.0800 ms	0.0681 ms	0.1574 ms	0.4982 ms
VoIP Call	19.9290 ms	19.8409 ms	40.0373 ms	8.7081 ms

As far as for the file interception, if one compares the KPIs presented in Table 3.3 related to the LI implementation in the 5G NR (Section 3.2.1) and in 5GC (Section 3.1.1), it seems that the theorized and hoped-for improvement in network performance, in terms of latency of the LI procedures, does not occur. Indeed, the average End-to-end LI latency increases of about 0.3 ms. However, this makes sense according to the OpenLI specifications, which state that the correct functioning of a POI in the presence of high data rates (which happens in the case of file transfer, while it does not occur during a VoIP call), occurs when 16 GB of RAM and 8 dedicated CPUs are allocated [26]. Obviously, given the small size of the considered files, excellent results are still obtained when the proposed LI framework is deployed in the 5GC. Conversely, since the entire 5G network infrastructure runs on the same workstation, albeit on a standalone container, it is

not possible to allocate the computational capacity the gNB would need to host a Point of Interception. In contrast, despite the hardware constraints, the container handling Core Network packet switching (i.e. the UPF) has no difficulty in performing both functions. Consequently, as can be seen from a comparison of Figures 3.1 and 3.11 concerning packet-wise latency at the POI, there is a doubling of the delay, due to the fact that the gNB_2 acts as a bottleneck.

On the other hand, by considering VoIP call scenarios, as still evident from Table 3.3 or the comparison between Figures 3.5 and 3.15 regarding packet-wise latency at the POI, this bottleneck phenomenon does not occur, since the gNB_2 has sufficient computational capacity to accommodate LI procedures in parallel with the forwarding of packets to and from the UE_2. In this scenario, since the VoIP session implies a largely lower data rate, it is evident how the shift of LI procedures from the Core to the Edge of the network matches the theoretical expectations, resulting in a consistent improvement of End-to-end LI Latency performance per packet, which becomes 4.5 times lower. Furthermore, there is still a negligible impact of LI procedures on the QoS perceived by the user, as illustrated in Section 3.2.2.

In addition, the data in Table 3.3 regarding the average UPF processing time per packet, when the system is in steady state, highlight how the migration of the LI workload at the edge of the network leads to a lower latency per packet, translating it to a faster processing of packets in the 5GC.

In conclusion, it is possible to state that the cutting-edge LI implementation solution proposed in this thesis, which extends the work already presented by the authors in the study in [14], carries a considerable advantage in terms of interception latency, with the possibility of intercepting communications at the edge of the network.

CONCLUSIONS

In this thesis, the topic of Lawful Interception (LI) in the new 5G telecommunication networks was thoroughly investigated, pointing out the problems and criticalities raised by the European Union in terms of national security, caused by the widespread adoption of robust end-to-end encryption standards. In response to this urgent need, a 3GPP-compliant Lawful Interception framework was developed and described.

The proposed LI framework is able, through the implementation of a key escrow system, not only to capture and transmit end-to-end encrypted traffic to a LEA, but also to provide a real-time service that does not impact the perceived QoS of the users subject to a warrant.

Recalling the study conducted, Chapter 1 provided a theoretical overview concerning the architecture, protocol stack and security of 5G, as well as giving an illustration of the 3GPP standard concerning the entities involved in a lawful interception framework in fifth-generation networks. This state-of-the-art presentation paved the way for presenting and justifying the issues represented by end-to-end encryption for national security and introducing the reader to key escrow procedures.

Based on these premises, the Chapter 2 detailed the software implementation of the proposed LI framework, presenting the tools used, the organization and configuration of each entity participating in the end-to-end communication and in the interception process that allows the delivery of IRIs and CCs to the LEA. In detail, two implementation designs of the proposed LI framework were analyzed, during encrypted end-to-end VoIP communications or file sharing: in the first scenario the POI is placed in the UPF and the LI procedures are the responsibility of the 5GC, while in the

second, cutting-edge solutions are considered. Specifically, the aim of this study was to describe how it is possible and advantageous to implement the proposed LI framework at the network edge, deploying the POI in the 5G NR.

Finally, in order to objectively validate the two implementations, Chapter 3 described the extensive number of simulations conducted by involving interception of files having four different sizes and four VoIP calls with different durations. For each file or VoIP call, 10^2 simulations were performed, the results of which were processed to evaluate the performance of the proposed LI framework, in terms of end-to-end LI latency per packet and the impact of LI operations on the quality of service perceived by the user. This last chapter has discussed how the already excellent performance obtained by deploying the proposed LI framework in the 5GC, can be further enhanced if the Multi-Edge Computing (MEC) paradigm is exploited, leading to significantly improved performances.

Future developments of this work, as previously pointed out, will aim towards a separation of the hardware for the execution of the 5GC, 5G NR and LI services, in order to refine the results of this thesis and extend them to more complex and advanced scenarios (e.g. with higher data rates and with multiple users intercepted in parallel). Furthermore, the implementation of the proposed LI framework within multiple network slices will be investigated.

APPENDIX

End-to-End encryption and Key Escrow algorithm

The encryption algorithm used to perform the Key Escrow procedure, enabling the LEA to retrieve the session key related to the user and the communication for which a warrant has been issued, is the IDBC Algorithm [29]. To explain the mathematical workings of this algorithm, let us suppose, as in the simulated scenarios, there is a communication in progress between UE_1 and UE_2, both connected to the same 5G infrastructure and in cells managed by two different gNBs. The entire key escrowing process and the entities involved are illustrated in Figure 21.

The first phase of the IDBC algorithm is called the *key negotiation phase*. In this preliminary phase, the TKA, possessing a master secret key $M \in \mathcal{Z}_p{}^*$, securely generates and distributes pairs of public and private keys to the two users, along with the symmetric keys k_1 and k_2, which UE_1 and UE_2 respectively use to secure their communication with the AUSF. Moreover, the two UEs randomly generates the two secret session nounces r_1 and r_2. Then, assuming that UE_1 and UE_2 have previously shared their unique identities (ID_1 and ID_2) with the TKA, the TKA uses a hash function $\mathcal{H} : \mathcal{Z}_p^* \to \mathcal{G}$ to generate:

$$\begin{cases} p_1 = \mathcal{H}(ID_1) & \text{UE\_1 public key,} \\ P_1 = M\mathcal{H}(ID_1) & \text{UE\_1 private key,} \end{cases}$$

and

$$\begin{cases} p_2 = \mathcal{H}(ID_2) & \text{UE\_2 public key,} \\ P_2 = M\mathcal{H}(ID_2) & \text{UE\_2 private key,} \end{cases}$$

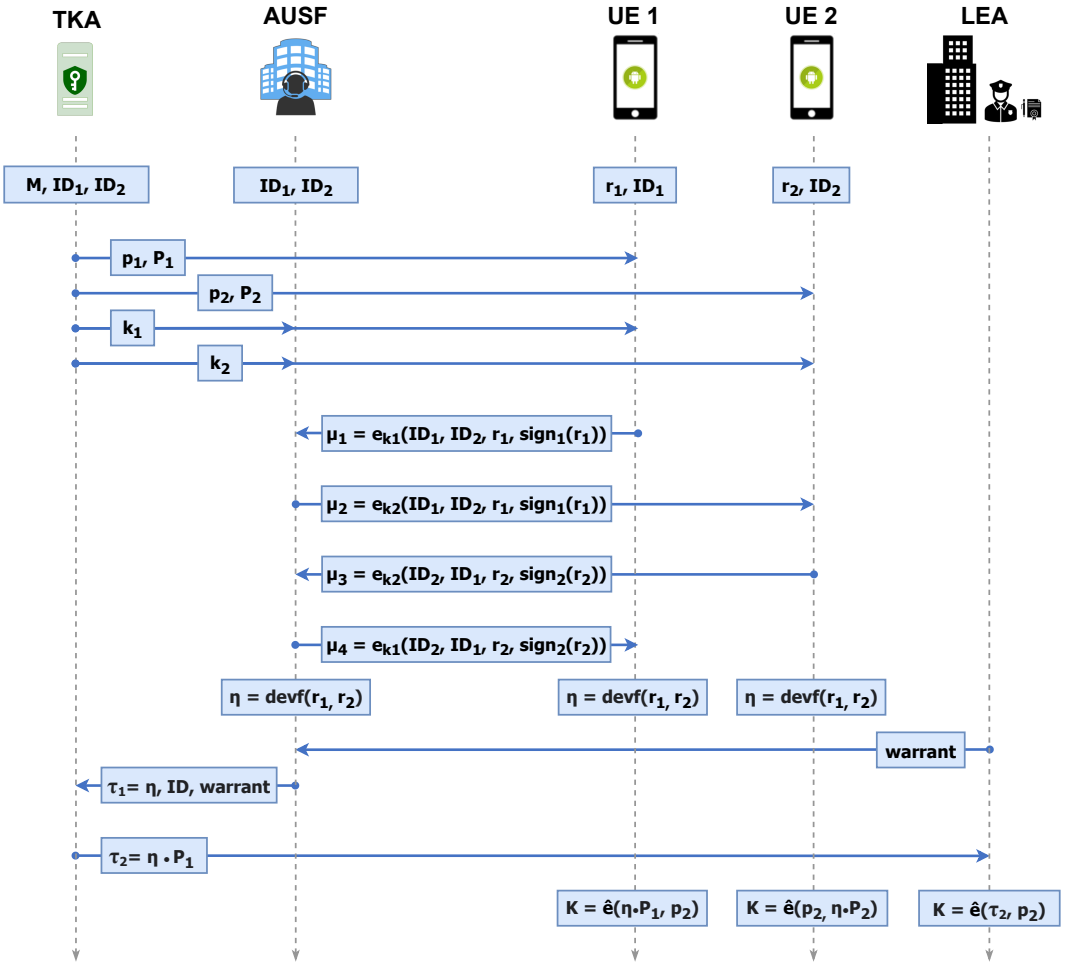


Figure 21: IDBC Algorithm's phases for key negotiation and key escrow.

where the computing complexity of obtaining M is equivalent to that of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP). After this setup phase, the UE_1 sends μ_1 to the AUSF:

$$\mu_1 = E_{k_1}(ID_1 \,||\, ID_2 \,||\, r_1 \,||\, sign_1(r_1)),$$

where r_1 is a random number produced by the UE_1, E_{k_1} indicates the encryption function using the shared key k_1, and $sign_1(r_1)$ is the matching signature. As a result, after receiving and decoding μ_1, the AUSF creates μ_2 to be transmitted to UE_2 and confirms r_1 using the signature $sign_1(r_1)$.

$$\mu_2 = E_{k_2}(ID_1 \,||\, ID_2 \,||\, r_1 \,||\, sign_1(r_1)),$$

where E_{k_2} indicates the encryption function using the shared key k_2, and $sign_1(r_1)$ is the signature associated to the nounce r_1.

After μ_1 and μ_2 being exchanged, UE_2 decrypts it and uses the signature $sign_1(r_1)$ to confirm r_1. Then, it sends μ_3 to the AUSF:

$$\mu_3 = E_{k_2}(ID_2 \,||\, ID_1 \,||\, r_2 \,||\, sign_2(r_2)),$$

where, E_{k_2} denotes the encryption function adopting the shared key k_2 and $sign_2(r_2)$ is the signature corresponding to the UE_2's nounce r_2.

After validating μ_3, the AUSF generates μ_4 and transmits it to the UE_1 after validating μ_3.

$$\mu_4 = E_{k_1}(ID_2 \,||\, ID_1 \,||\, r_2 \,||\, sign_2(r_2)),$$

where, E_{k_2} denotes the encryption function adopting the shared key k_2. By verifying $sign_2(r_2)$ now the UE_1 is now able to verify the UE_2 identity, the AUSF and both UEs are mutually authenticated in pairs. Therefore, UE_1, UE_2 and the AUSF are able to compute $\eta = devf(r_1, r_2)$, where $devf$ is a derivation function from r_1 and r_2.

Once that the communication procedure within the subscribers is completed, when the LEA sends a valid warrant to the AUSF via the HI1 interface, the AUSF computes τ_1 (containing the warrant) and transmits it to the TKA:

$$\tau_1 = \eta \,||\, ID_1 \,||\, \text{warrant}\,.$$

Consequently, through the HI2 interface, the TKA can communicate τ_2 to the LEA:

$$\tau_2 = \eta \cdot M\mathcal{H}(ID_1),$$

where the $\cdot$ operator denotes the multiplication. Now each entity possesses the necessary cryptographic material to independently produce k_{12}, the communication session key that will be used to encrypt the end-to-end communication session:

1. the UE_1 computes $k_{12} = \hat{e}\,(\eta \cdot M\mathcal{H}(ID_1), \mathcal{H}(ID_2))$.
2. the UE_2 computes $k_{21} = \hat{e}\,(\mathcal{H}(ID_1), \eta \cdot M\mathcal{H}(ID_2))$.

3. the LEA, by using the public hash function $\mathcal{H} : \mathcal{Z} \rightarrow \mathcal{P}$, calculates $\mathcal{H}(ID_1)$ and $\mathcal{H}(ID_2)$, and with τ_2 correctly computes the communication session key k_{12}.

In this case, the bilinear function operation is defined by $\hat{e}(\cdot)$. The validity of the two equations is demonstrated using the IDBC-based model features given in [29]:

$$\begin{aligned} k_{12} &= \hat{e}\left(\eta \cdot M\mathcal{H}(ID_1),\, \mathcal{H}(ID_2)\right) = \\ &= \hat{e}\left(\mathcal{H}(ID_1),\, \mathcal{H}(ID_2)\right)^{\eta \cdot M} = \\ &= \hat{e}\left(\mathcal{H}(ID_1),\, \eta \cdot M\mathcal{H}(ID_2)\right) = k_{21}\ . \end{aligned}$$

BIBLIOGRAPHY

[1] 3rd Generation Partnership Project (3GPP), "Technical Specification Group Services and System Aspects; Security; Lawful Interception requirements; (Release 19)," 3rd Generation Partnership Project (3GPP), Tech. Rep. TS 33.126, 2024. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/33_series/33.126/33126-j00.zip.

[2] 3rd Generation Partnership Project (3GPP), "Technical Specification Group Services and System Aspects; Security; Lawful Interception (LI) architecture and functions; (Release 18)," 3rd Generation Partnership Project (3GPP), Tech. Rep. TS 33.127, 2024. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/33_series/33.127/33127-i70.zip.

[3] 3rd Generation Partnership Project (3GPP), "Technical Specification Group Services and System Aspects; Security; Protocol and procedures for Lawful Interception (LI); (Release 18)," 3rd Generation Partnership Project (3GPP), Tech. Rep. TS 33.128, 2024. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/33_series/33.128/33128-i70.zip.

[4] ETSI. "5G Technology," European Telecommunications Standards Institute. (n.d.), [Online]. Available: https://www.etsi.org/technologies/5g.

[5] 3GPP. "5G System Overview," 3rd Generation Partnership Project. (2022), [Online]. Available: https://www.3gpp.org/technologies/5g-system-overview.

[6] 3rd Generation Partnership Project (3GPP), "Technical Specification Group Services and System Aspects; Network architecture (Release 18)," 3rd Generation Partnership Project (3GPP), Tech. Rep. TS 23.002, 2024. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/23_series/23.002/.

[7] 3rd Generation Partnership Project (3GPP), "Technical Specification Group Services and System Aspects; System Architecture for the 5G System; (Release 18)," 3rd Generation Partnership Project (3GPP), Tech. Rep. TS 23.501, 2024. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/23_series/23.501/23501-i50.zip.

[8] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider, "Nfv and sdn—key technology enablers for 5g networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2468–2478, 2017. DOI: 10.1109/JSAC.2017.2760418.

[9] F. Rinaldi, A. Raschellà, and S. Pizzi, "5g nr system design: A concise survey of key features and capabilities," *Wireless Networks*, vol. 27, no. 8, pp. 5173–5188, 2021, ISSN: 1572-8196. DOI: 10.1007/s11276-021-02811-y. [Online]. Available: https://doi.org/10.1007/s11276-021-02811-y.

[10] 3rd Generation Partnership Project (3GPP), "Technical Specification Group Services and System Aspects; Security architecture and procedures for 5G system; (Release 18)," 3rd Generation Partnership Project (3GPP), Tech. Rep. TS 33.501, 2024. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/33_series/33.501/33501-i50.zip.

[11] S. Behrad, E. Bertin, and N. Crespi, "Securing authentication for mobile networks, a survey on 4g issues and 5g answers," in *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, 2018, pp. 1–8. DOI: 10.1109/ICIN.2018.8401619.

[12] A. R. Prasad, S. Arumugam, B. Sheeba, and A. Zugenmaier, "3gpp 5g security," *Journal of ICT Standardization*, vol. 6, no. 1-2, pp. 137–158, 2018.

[13] J. Lee, H. Kim, C. Park, Y. Kim, and J.-G. Park, "Ai-based network security enhancement for 5g industrial internet of things environments," in *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*, 2022, pp. 971–975. DOI: 10.1109/ICTC55196.2022.9952490.

[14] I. Huso, M. Olivieri, L. Galgano, A. Rashid, G. Piro, and G. Boggia, "Design and implementation of a looking-forward lawful interception architecture for future mobile communication systems," *Computer Networks*, vol. 249, p. 110 518, 2024, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2024.110518. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128624003505.

[15] Council of the European Union, *Presidency Law Enforcement Working Party Position paper on 5G by Europol*, EU Council Document, Note No. 8268/19, Brussels, 2019. [Online]. Available: https://www.statewatch.org/media/documents/news/2019/jun/eu-council-europol-position-paper-5g-8268-19.pdf.

[16] Council of the European Union, *Law enforcement and judicial aspects related to 5G*, EU Council Document, Note No. 8983/19, Brussels, May 2019. [Online]. Available: https://data.consilium.europa.eu/doc/document/ST-8983-2019-INIT/en/pdf.

[17] M. Alatawi and N. Saxena, "Sok: An analysis of end-to-end encryption and authentication ceremonies in secure messaging systems," in *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '23, Guildford, United Kingdom: Association for Computing Machinery, 2023, pp. 187–201, ISBN: 9781450398596. DOI: 10.1145/3558482.3581773. [Online]. Available: https://doi.org/10.1145/3558482.3581773.

[18] G. Grover, T. Rajwade, and D. Katira, "The ministry and the trace: Subverting end-to-end encryption," *NUJS Law Review*, vol. 14, p. 223, 2021.

[19] S. Cha, S. Baek, and S. Kim, "Blockchain based sensitive data management by using key escrow encryption system from the perspective of supply chain," *IEEE Access*, vol. 8, pp. 154 269–154 280, 2020. DOI: 10.1109/ACCESS.2020.3017871.

[20] P. Kühn, P. Imperatori, and C. Reuter, "U.s. security policy: The dual-use regulation of cryptography and its effects on surveillance," *European Journal for Security Research*, vol. 7, no. 1, pp. 39–65, Jul. 2022, ISSN:

2365-1695. DOI: 10.1007/s41125-022-00080-0. [Online]. Available: https://doi.org/10.1007/s41125-022-00080-0.

[21] S. M. Bellovin, J. Benaloh, M. Blaze, *et al.*, "The risks of key recovery, key escrow, and trusted third-party encryption,"

[22] Open5GS Website. (), [Online]. Available: https://open5gs.org/.

[23] herlesupreeth. docker_open5gs Git Hub Repository. (), [Online]. Available: https://github.com/herlesupreeth/docker_open5gs/.

[24] Docker Website. (), [Online]. Available: https://www.docker.com/.

[25] UERANSIM Git Hub repository. (), [Online]. Available: https://github.com/aligungr/UERANSIM/.

[26] OpenLI Website. (), [Online]. Available: https://www.openli.nz/.

[27] Asterisk Website. (), [Online]. Available: https://www.asterisk.org/.

[28] PJSIP Project Online Documentation. (), [Online]. Available: https://cods.pjsip.org/en/latest/index.html/.

[29] K. Han, C. Y. Yeun, T. Shon, J. Park, and K. Kim, "A scalable and efficient key escrow model for lawful interception of IDBC-based secure communication," *International Journal of Communication Systems*, vol. 24, no. 4, 2011.