ETSI TR 133 727 V19.0.0 (2026-07)

TECHNICAL REPORT

5G;
Lawful Interception Handling of Protected Services
(3GPP TR 33.727 version 19.0.0 Release 19)

Reference
DTR/TSGS-0333727vj00

Keywords
5G

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables (European Standard (EN), Technical Specification (TS), Group Specification (GS) or ETSI Standard (ES)) may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI IPR online database.

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Legal Notice

This Technical Report (TR) has been produced by ETSI 3rd Generation Partnership Project (3GPP).

The present document may refer to technical specifications or reports using their 3GPP identities. These shall be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between 3GPP and ETSI identities can be found at 3GPP to ETSI numbering cross-referencing.

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Contents

Foreword

This Technical Report has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

In the present document, modal verbs have the following meanings:

shall indicates a mandatory requirement to do something

shall not indicates an interdiction (prohibition) to do something

The constructions "shall" and "shall not" are confined to the context of normative provisions, and do not appear in Technical Reports.

The constructions "must" and "must not" are not used as substitutes for "shall" and "shall not". Their use is avoided insofar as possible, and they are not used in a normative context except in a direct citation from an external, referenced, non-3GPP document, or so as to maintain continuity of style when extending or modifying the provisions of such a referenced document.

should indicates a recommendation to do something

should not indicates a recommendation not to do something

may indicates permission to do something

need not indicates permission not to do something

The construction "may not" is ambiguous and is not used in normative elements. The unambiguous constructions "might not" or "shall not" are used instead, depending upon the meaning intended.

can indicates that something is possible

cannot indicates that something is impossible

The constructions "can" and "cannot" are not substitutes for "may" and "need not".

will indicates that something is certain or expected to happen as a result of action taken by an agency the behaviour of which is outside the scope of the present document

will not indicates that something is certain or expected not to happen as a result of action taken by an agency the behaviour of which is outside the scope of the present document

might indicates a likelihood that something will happen as a result of action taken by some agency the behaviour of which is outside the scope of the present document

might not indicates a likelihood that something will not happen as a result of action taken by some agency the behaviour of which is outside the scope of the present document

In addition:

is (or any other verb in the indicative mood) indicates a statement of fact

is not (or any other negative verb in the indicative mood) indicates a statement of fact

The constructions "is" and "is not" do not indicate requirements.

1 Scope

Offering communication services with end-to-end (E2E) cryptographic protection is becoming more desirable and more common. 3GPP-defined services (such as those of the IMS framework) have so far largely relied on trustworthy network infrastructure, complemented by strong cryptographic access security and hop-by-hop protection in the core and service domains. However, also here, E2E protected services are gaining attractivity. Indeed, the solution defined in TS 33.535, Authentication and Key Management for Applications based on 3GPP credentials in the 5G System (AKMA), can be leveraged as a basis for protecting basically any IP-based service provided to a UE (an ME with USIM-credentials). While this trend is both natural and desirable from a user privacy point of view, it creates problems related to regulatory obligations for CSPs to provide Lawful Interception (LI). The problems are particularly emphasised in roaming scenarios, where each of two different CSPs in distinct jurisdictions might independently need to provide LI without reliance on co-operation by the other CSP and/or law enforcement in the other jurisdiction.

The currently dominant approach to tackle these issues has been to simply not activate the protection (encryption) when it potentially conflicts with LI requirements. Going forward, this is most likely not a sustainable solution. For example, it potentially exposes the traffic to any 3rd party and, moreover, it typically also results in data integrity being disabled, while the issues are predominantly tied to data confidentiality.

The present document provides a technical study on how to enable protected services without negative consequences for LI. The CSP's LI obligation to provide communication with the encryption removed only applies when the CSP also provides the technical means enabling the protection (e.g. key management), and therefore the scope is limited to that use case.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

[2] 3GPP TS 33.107: "Lawful interception architecture and functions".

[3] 3GPP TS 33.126: "Lawful Interception requirements".

[4] 3GPP TS 33.127: "Lawful Interception (LI) architecture and functions".

[5] 3GPP TS 33.128: "Protocol and procedures for Lawful Interception (LI)".

[6] 3GPP TS 33.163: "Battery Efficient Security for very low throughput Machine Type Communication (MTC) devices (BEST)".

[7] 3GPP TS 33.203: "3G security; Access security for IP-based services".

[8] 3GPP TS 33.220: "Generic Authentication Architecture (GAA); Generic Bootstrapping Architecture (GBA)".

[9] 3GPP TS 33.222: "Generic Authentication Architecture (GAA); Access to network application functions using Hypertext Transfer Protocol over Transport Layer Security (HTTPS)".

[10] 3GPP TS 33.501: "Security architecture and procedures for 5G System".

[11] 3GPP TS 33.535: "Authentication and Key Management for Applications (AKMA) based on 3GPP credentials in the 5G System (5GS)".

[12] 3GPP TR 33.828: "IP Multimedia Subsystem (IMS) media plane security".

[13] IETF RFC 3711: "The Secure Real-time Transport Protocol (SRTP)".

[14] IETF RFC 5246: "The Transport Layer Security (TLS) Protocol Version 1.2".

[15] IETF RFC 8446: "The Transport Layer Security (TLS) Protocol Version 1.3".

[16] IETF RFC 9147: "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3".

3 Definitions of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in 3GPP TR 21.905 [1] and the following apply. A term defined in the present document takes precedence over the definition of the same term, if any, in 3GPP TR 21.905 [1].

The study of the present document is concerned with providing a technical description of how to process, as part of Lawful Interception, a flow of security protected PDUs, transmitted across a CSP network, where said PDUs carry UP traffic related to some service. Specifically, the processing is carried out at a point inside the CSP network, using cryptographic transforms such as decryption, which would otherwise only take place at the receiving endpoint. To make the description of this processing unambiguous and as simple as possible, the following terms are defined.

Auxiliary security parameters: A set of security related parameters, other than cryptographic keys, associated with a security protocol and needed to perform security processing according to said security protocol.

EXAMPLE: Selected encryption algorithm, nonces, and synchronization information such as sequence numbers are typical examples of auxiliary security parameters.

Cryptographic context: Information maintained locally at the sender/receiver and used as basis to determine how to cryptographically process each PDU. This information includes cryptographic keys and auxiliary security parameters.

Cryptographic Synchronization Information (CSI): The complete set of information used (together with cryptographic keys) as input to the cryptographic processing of each PDU at the sender/receiver. CSI is formed by a combination of information from the cryptographic context and explicit cryptographic synchronization information (see below).

CSP-provided keys: Cryptographic keys, generated by technical means provided by a CSP, where said keys are used to encrypt traffic carrying services provided to a subscriber of the CSP. Due to LI obligations of the CSP, the key generation is done in a way which enables the CSP to generate the same keys.

EXAMPLE: The "technical means" can comprise USIM, eSIM, a key management server, some application provided by the CSP and executed in the UE, or combinations thereof.

Decryption Point-of-Intercept (D-POI): A CC-POI inside the CSP network which processes encrypted xCC, decrypting it to produce plaintext xCC or plaintext CC.

NOTE: Whether the D-POI produces xCC or CC depends on its architectural placement, for which options will be discussed elsewhere in the present document.

Explicit Cryptographic Synchronization Information (ECSI): Part of CSI carried in-band, as part of, or derived from, security protocol PDUs.

Perfect forward secrecy: Property of a security protocol by which the compromise of keys involved in a particular session does not negatively affect the security of prior sessions. If the compromised keys are only of temporary nature, also future sessions are protected from adverse effects.

Security protocol: A protocol used between a sender and a receiver to provide security services such as data confidentiality and/or data integrity to communication. The security protocol receives an SDU from higher layer and produces security protocol PDUs, transmitted over lower layers. In the present document, the sender and receiver roles typically correspond to a UE and some application layer service, respectively. However, it is also possible that both the sender and receiver are UEs.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in 3GPP TR 21.905 [1] and the following apply. An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in 3GPP TR 21.905 [1].

AAnF	AKMA Anchor Function
ADMF	Administration Function
AF	Application Function
AKID	AKMA Key Identifier
AKMA	Authentication and Key Management for Applications
BBIFF	Bearer Binding Intercept and Forward Function
CC	Content of Communication
CID	Connection Identifier
CSI	Cryptographic Synchronization Information
CSP	Communication Service Provider
DN	Data Network
DSF	Decryption Support Function
D-POI	Decryption Point of Intercept
E2E	End-to-end
E2EE	End-to-end Encryption
ECSI	Explicit Cryptographic Synchronization Information
FQDN	Fully Qualified Domain Name
GBA	Generic Bootstrapping Architecture
HPLMN	Home PLMN
IMS	IP Multimedia Subsystem
IRI	Intercept Related Information
IV	(cryptographic) Initialization Value
KDF	Key Derivation Function
KID	AKMA Key Identifier
K_{LI}	Decryption key(s) for services encrypted by CSP-provided keys
KSF	Key Server Function
LEA	Law Enforcement Agency
LEMF	Law Enforcement Monitoring Facility
LI	Lawful Interception
LICF	Lawful Interception Control Function
LI_HI1	LI_Handover Interface 1
LI_HI2	LI_Handover Interface 2
LI_HI3	LI_Handover Interface 3
LIPF	LI Provisioning Function
LI_T1	Lawful Interception Triggering Interface 1
LI_T3	Lawful Interception Triggering Interface 3
LI_TSH	Lawful Interception Triggering Interface for security protocol handshakes
LI_X1	Lawful Interception Internal Interface 1
LI_X2	Lawful Interception Internal Interface 2
LI_X3	Lawful Interception Internal Interface 3
LI_X1_CR	Lawful Interception Internal Interface for provisioning of cryptographic parameters
LI_X3_CR	Lawful Interception Internal Interface for cryptographic parameters captured from user plane
LMSSF	LI Mirror Security State Function
LMSSF-C	LI Mirror Security State Function Control
LMSSF-S	LI Mirror Security State Function Storage
MAC	Message Authentication Code
MDF	Mediation and Delivery Function
MDF2	Mediation and Delivery Function 2
MDF3	Mediation and Delivery Function 3
NF	Network Function
PDU	Protocol Data Unit

PLMN	Public Land Mobile Network
POI	Point Of Interception
PSK	Pre-Shared Key
SCAS	Security Assurance Specification
SDU	Service Data Unit
SHIFF	Security Handshake Interception and Forwarding Function
SMF	Session Management Function
SPDF	Security Protocol Detection Function
SRTP	Secure Real-time Transport Protocol
STF	Security Terminating Function
SUPI	Subscriber Permanent Identifier
TF	Triggering Function
UE	User Equipment
UPF	User Plane Function
VPLMN	Visited PLMN
xCC	LI_X3 Communications Content
xIRI	LI_X2 Intercept Related Information

4 Background

4.1 General

3GPP has currently defined two general frameworks that allow subscriber credentials (the USIM) to be utilized to derive cryptographic keys for essentially any type of IP-based service. The two frameworks are the Generic Bootstrapping Architecture (GBA, TS 33.220 [8]) and Authentication and Key Management for Applications based on 3GPP credentials in the 5G System (AKMA, TS 33.535 [11]). There also exist service-specific security solutions that rely on USIM, e.g. the Battery Efficient Security for very low throughput Machine Type Communication devices (BEST, TS 33.163 [6]) and the Access security for IP-based services (IMS signalling security, TS 33.203 [7]), the latter with media plane security extensions described in TR 33.828 [12].

In all of these cases, a situation can occur resulting in that traffic is protected (including encryption) in a more network-wide fashion (potentially end-to-end), in turn potentially leading to conflicts with fulfilling regulatory obligations with respect to Lawful Interception. If a CSP initiates encoding, compression, or encryption of telecommunications traffic, LEAs, depending on regional or national regulatory requirements, generally require the CSP to provide intercepted communications "en clair". Such requirements are observed in the 3GPP specifications, but until recently, mainly by expressing recommendations of the *use* of encryption.

EXAMPLE: As stated in [6], Annex A: "Due to regulatory requirements, operators may have to disable the BEST service for UEs roaming in their network".

This approach is not optimal since it in practice leads to disabling of the traffic protection altogether (including data encryption, data integrity, and data origin authentication).

4.2 Current LI Architecture

4.2.1 Overview

As of 3GPP Release 17, the Lawful Interception architecture of TS 33.127 [4] has been extended by normative specifications describing a general LI-framework for services encrypted by CSP-provided keys and a more specific solution for the case that the service is protected (encrypted) based on AKMA keys (see clause 7.15 of TS 33.127 [4]).

To this end, the architectural framework of TS 33.127 [4] defines the notion of a *Security Termination Function* (STF). This corresponds to the NF providing some service to the UE (messaging, chat, etc.) and where the service is encrypted by CSP-provided keys. The encryption is applied end-to-end, between the UE and the STF. The CSP-provided keys, in turn, are provided by a *Key Server Function* (KSF), located at, and operated by the CSP.

EXAMPLE: A concrete instantiation of an STF and KSF can be exemplified with the KSF corresponding to an AKMA Anchor Function (AAnF) and the STF corresponding to an AKMA Application Function (AF) according to TS 33.535 [11].

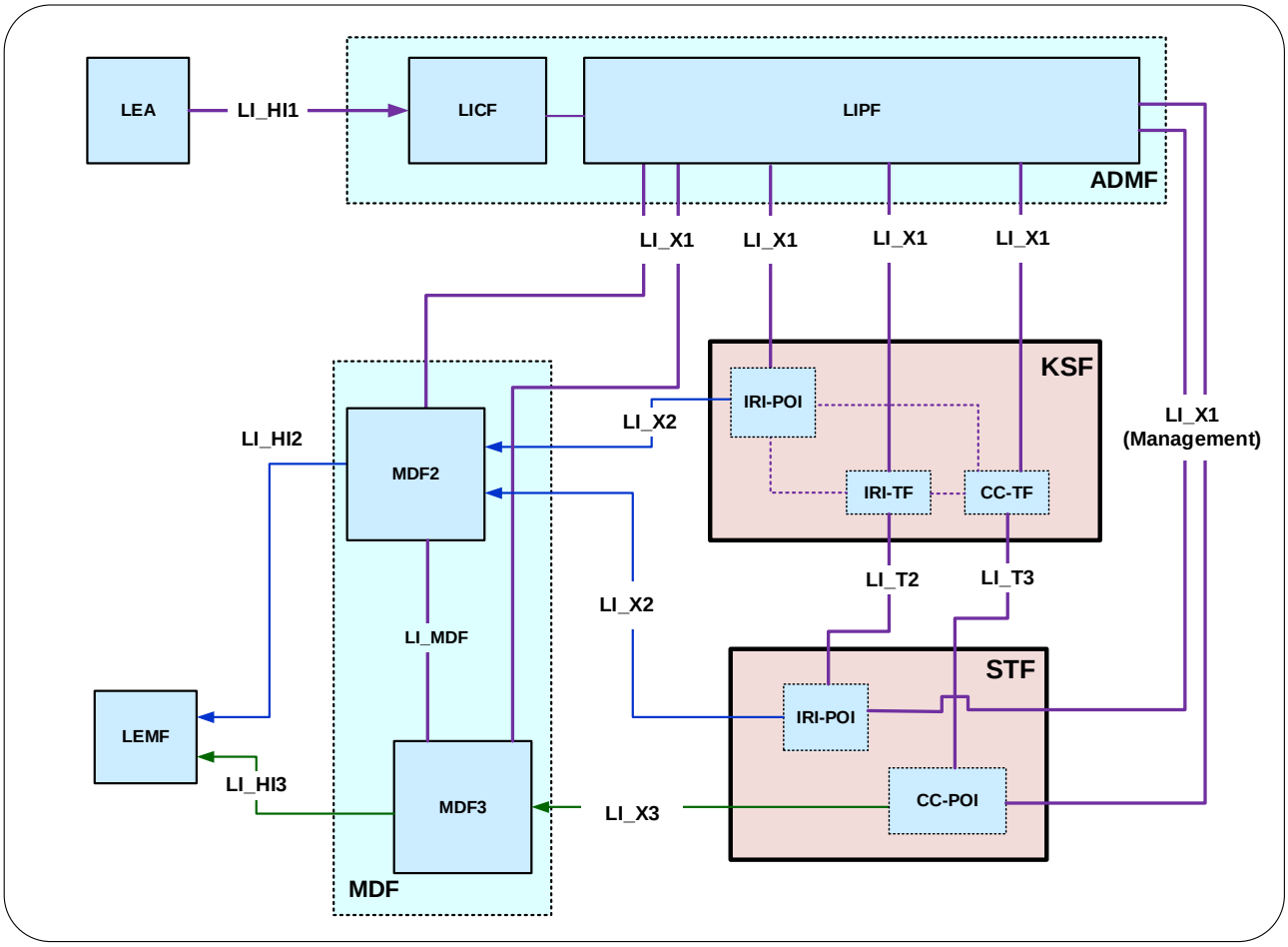

Figure 4.2.1-1: General (non-roaming) LI-architecture for CSP-provided keys according to TS 33.127. If the STF uses 5G-native identifiers, the triggering of POIs in the STF from the IRI-TF and CC-TF in the KSF could be replaced by direct provisioning of the STF using LI_X1, see text.

4.2.2 Roaming

The current LI-architecture of TS 33.127 [4] covers only the non-roaming case. Technical requirements can be found in TS 33.126 [3].

4.2.3 LI Provisioning and Triggering

The KSF is provisioned based on SUPI. If also the STF provides services based on a 5G-native identifier such as SUPI, standard LI provisioning over LI_X1 is possible also for the STF. If some other identifier is used by the STF, it might be necessary to instead use triggering: determination of a subscriber as being an LI target might not be possible until the STF actively requests key material from the KSF where triggering functions of the KSF maps the request to a SUPI and triggers the STF.

4.2.4 Generation of IRI

The scope of the LI-solution defined in TS 33.127 [4] is to collect IRI from the KSF (e.g. CSP-provided keys) and optionally from the STF over LI_X2. IRI collected from the STF can comprise further keys and/or other auxiliary security parameters such as the selected encryption protocol or algorithm.

The IRI collected from the KSF comprises at least key material which forms the basis for the actual encryption/decryption of the services, between the STF and a UE. This key material is generically referred to as K_{LI}. The key K_{LI} is associated with a specific subscriber and could also be tied to a specific (subscriber, STF)-pair. In the latter case, K_{LI} is derived from a user-specific key K_{USER}. There is an identifier associated with K_{USER} which uniquely identifies the K_{USER}, at least within the CSP and this identifier is generically denoted KID.

When the UE requests protected services from an STF, the request typically starts with a security handshake, establishing further IRI related to the protection. For key management protocols such as AKMA, the STF might not always be located within the CSP domain. Thus, in order to make the discussion as generic as possible, the present document makes no assumption on the ability to collect any IRI whatsoever from the STF. As a consequence, collection of IRI from in-band signalling between UE and STF becomes an identified key issue (see clause 5.1.3).

If AKMA is used, a mapping from TS 33.127 [4] to TS 33.535 [11] can be made according to table 4.2.4-1.

Table 4.2.4-1: Mapping functions between the general architecture and AKMA

Term in the present document and TS 33.127	AKMA correspondence	Reference
KSF	AAnF	TS 33.535 [11] clause 4.2.1
STF	AF	TS 33.535 [11] clause 4.2.2
K_{USER}	K_{AKMA}	TS 33.535 [11] clause 6.1
K_{STF}	K_{AF}	TS 33.535 [11] clause 6.2
K_{LI}	K_{AKMA} and/or K_{AF}	TS 33.535 [11] clause 6.1, 6.2
Key identifier, KID	A-KID	TS 33.535 [11] clause 4.4.2
Auxiliary security parameters	Ua* security protocol parameters	TS 33.535 [11] clause 4.4.1

NOTE 1: In almost all cases, the encryption/decryption key used is distinct in dependence on whether the traffic is sent from UE to the STF, or, from STF to UE. Both of these keys are usually derived from the same K_{LI} and appropriate notation will be introduced later to unambiguously refer to the two directional keys when this is relevant.

NOTE 2: For the purposes of the present document, the security protocol applied to UP traffic between the STF and UE can be assumed to use *symmetric cryptography* so that the terms "encryption key" and "decryption key" refer to one and the same key.

4.2.5 Generation of CC

In case the STF provides a 3GPP-defined service operated by the CSP, the STF might further be equipped with a CC-POI. If this is the case, there is in general no need to use the collected IRI to explicitly perform decryption of the LI product since a CC-POI can then be assumed to reside "behind" the decryption functionality of the STF. However, since the present document makes no assumption on LI functionality at the STF, this case is not discussed further. Instead, defining an appropriate CC-POI functionality within the CSP network is defined as one of the key issues (see clause 5.4.1).

4.3 Cryptographic processing for protected services

4.3.1 General

The processing that needs to be performed within the CSP's LI system to convert confidentiality protected (encrypted) user plane PDUs into plaintext service-related communication content is for obvious reasons very similar to the processing that takes place at the receiving endpoint. This, in turn, mirrors processing at the sender.

EXAMPLE: An encryption processing step at the sender will correspond to a decryption processing step at the receiver.

Therefore, to understand the problem space better, it is useful to first describe the security processing at the sender/receiver in terms of a hypothetical security protocol which describes the processing steps as generally as possible. Readers well acquainted with security protocols can without loss skip through the remainder of this clause.

All security protocols have a quite universally valid structure in terms of processing flow.

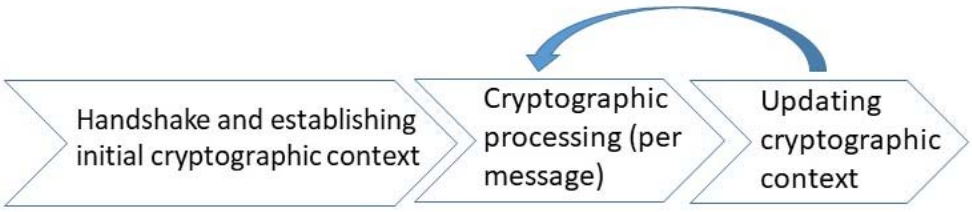

Figure 4.3.1-1: Security protocol processing overview.

4.3.2 Handshake and cryptographic context initialization

Most protocols first use a handshake phase which is used to (mutually) authenticate the communicating parties, and to initiate a cryptographic context to be used to protect a specific session. This involves creating the necessary session keys, negotiating which cryptographic algorithms to use, etc. In general, the handshake results in initializing a local cryptographic context (or state) at the sender and receiver. When CSP-provided key management is used, the CSP-provided keys can be used for authentication, and the session keys are then created in dependence of those CSP-provided keys. Whether (and how) such handshake occurs depends on whether the security protocol has integrated its own authentication and key management functionality or whether a separate authentication and key agreement protocol is used for that purpose.

EXAMPLE: The (D)TLS protocols [14,15,16] have the authentication and key management "built-in" as discussed above, while the Secure Real-time Transport Protocol (SRTP) [13] does not.

The handshake is often a "once-per-session" matter, though further handshake messages might occur during an established session, e.g. to update keys.

4.3.3 Per-message processing

4.3.3.1 General

After the handshake has been completed, the security protocol handles its main responsibility. At the receiver end, this corresponds to converting received security protocol PDUs into plaintext messages passed upward in the stack to the application. This per-message processing is the focus of the present clause, and a process flow is shown in figure 4.3.3.1-1 below.

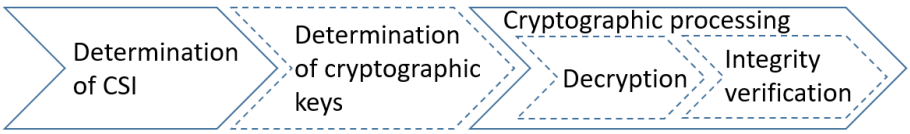

Figure 4.3.3.1-1: Security protocol processing (per message) on the receiving end.

NOTE: Since the present document focuses on the decryption process at within the CSP LI-system, processing at the sender end is largely omitted from the discussion.

The step "cryptographic processing" can be broken down into sub-steps "decryption" and "integrity verification" but the order of those two sub-steps can sometimes be reversed, and the two sub-steps can sometimes be integrated into a single cryptographic transform. The steps are described in more detail below.

4.3.3.2 Determination of CSI

In order to cope with possible unreliability of lower protocol layers (e.g. packet loss or reorder) and also to protect against malicious interference by an adversary, e.g. attempting to replay protected messages, each security protocol message (PDU) is processed relative to cryptographic synchronization information (CSI). This information can in part be carried in-band, denoted explicit cryptographic synchronization information (ECSI) and part of it can be implicit, as

part of, or derived from, the cryptographic context. The CSI serves the purpose of assigning a unique "identity" to each message/packet, allowing the receiver to reject replayed messages and to determine the position (ordering) of the message within the session. It is common to base the CSI on a sequence number or the like. This sequence number (or part thereof) is typically carried in-band when unreliable transport is used and can otherwise be part of the cryptographic context. For cryptographic reasons, another part of the CSI is often a random value which can be fixed for a session and/or be unique to each PDU.

EXAMPLE: Since TLS is likely to be commonly used, it is of interest to exemplify the CSI using this protocol. The CSI for decryption used in TLS always comprises an ECSI carried in the TLS record payload. For certain cipher suites (so called *AEAD-algorithms*, see clause 4.2.3.4) the CSI additionally makes use of an implicit component in the form of a per-session "nonce" (obtained from the handshake) and the 64-bit TLS sequence number (the two latter being part of the cryptographic context). The integrity verification in TLS always makes use of the 64-bit TLS sequence number for replay protection. Because TLS runs over TCP, there is no need to explicitly notify the TLS-layer of the next sequence number.

4.3.3.3 Determination of cryptographic keys

In most cases, the cryptographic keys used as input to cryptographic processing are static for the duration of the communication session and are stored as part of the cryptographic context. However, some notable examples to the contrary exist.

EXAMPLE 1: The SRTP protocol [13] has an option to deterministically refresh keys during a session: new keys are created after every r:th processed SRTP packet (r is a configurable parameter).

EXAMPLE 2: The (D)TLS 1.3 protocols (IETF 8446 [15] and IETF 9147 [16]) can start to transmit encrypted application data even before the handshake is complete and parts of the handshake can also be encrypted. In these cases, special keys are used for the "early data" and the handshake. New set of keys are taken into use after the handshake completes. These protocols can also update keys at any point in time by a special key update message.

4.3.3.4 Cryptographic message processing

As mentioned, the security protocol typically employs both data encryption and data integrity.

NOTE: From LI point of view, security protocols providing data integrity provides stronger means of attributing the intercepted traffic to a target's communication as only the target UE, the STF (or the CSP itself) could generate traffic that verifies correctly by knowledge of the keys. Integrity protection can also aid mid-session intercept as will be elaborated later in the present document.

Therefore, while the main focus of LI can be argued to be that of acquiring plaintext xCC, verifying integrity in the LI-system also brings benefits to the overall LI-product.

The integrity protection comprises adding a message authentication code (MAC) at the sender and to verify this at the receiver. To this end, there are four possibilities:

1. The security protocol first encrypts the application SDU, then adds integrity protection of the encrypted SDU (encrypt-then-MAC).
2. The security protocol encrypts the application SDU but adds integrity protection for the original unencrypted (plaintext) SDU (encrypt-and-MAC).
3. The security protocol adds integrity protection to the plaintext SDU, then encrypts *both* the plaintext SDU *and* the added message authentication code (MAC-then-encrypt).
4. A so-called *authenticated encryption* algorithm (AEAD) is used, which bundles data encryption with data integrity in (more or less) a single cryptographic transformation.

This maps to corresponding processing at the receiving end. In the first three cases, separate cryptographic keys are typically needed for encryption and data integrity while in the fourth case, a single key can be used. In the concrete examples based on the TLS-protocol described later in the present document, only approaches 3 and 4 occur.

As discussed, the communicating parties are also interested in protecting against replayed messages. This is logically part of the integrity verification step. To this end, the CSI is used. The CSI therefore, at least in part, often takes the

form of a sequence number, incrementing by "+1" on each message. When the receiver starts to process a new message, it first checks if a PDU with that sequence number has already been received (and verified). If so, the PDU is discarded as a potential replayed message. Otherwise, the integrity verification is performed. If the integrity verifies correctly, the receiver locally marks the corresponding message counter as received and otherwise discards the message. Depending on whether reliable transport is used or not, the replay protection scheme might need to be complemented with more elaborate window-based mechanisms, accepting messages whose sequence numbers lie within a certain window, relative to the previously received messages. Information about the window is stored in the cryptographic context. A more in-depth discussion is provided in annex A.

4.3.4 Updating cryptographic context

This step comprises updating the information in the cryptographic context in preparation for the next PDU. For example, updating the window of received PDUs, updating information relating to keys and/or CSI, etc.

4.4 Assumptions and principles

For the purpose of the present document, the following assumptions are made, in particular assumptions about the principal properties of the desired LI functionality. Examples and notes clarifying the rationale and impact of the assumptions and principles are also provided.

A0. Unless otherwise noted, the remainder of the present document uses the term "the CSP" to refer to the CSP for which LI obligations apply. In roaming, the CSP can correspond to the VPLMN or the HPLMN.

A1. The STF does not reside within the CSP and is not directly usable to collect further IRI.

EXAMPLE 1: In a non-roaming case, the STF can be located at an external DN.

EXAMPLE 2: In a roaming case, the STF could be located in the HPLMN or at an external DN.

A2. The key management is based on USIM (in the roaming case administrated by the home CSP) and follows some 3GPP specification, e.g. AKMA, TS 33.535 [11].

A3. The LI solution needs to have the capability of at least decrypting the UE-STF traffic.

A4. The LI solution is autonomous and does not depend on inter-CSP, LI-specific assistance.

NOTE 1: In the roaming case, full information on AKMA keys is in general only available in the HPLMN.

A5. The security (encryption) protocol in use between UE and STF also follows 3GPP specifications (possibly by cross-referencing a specification of some other SDO).

EXAMPLE 3: The CSP-provided keys alone could be insufficient to perform decryption, due to that the actual decryption key(s) depend also on protocol-specific signalling exchanged in-band between the UE and the STF. The CSP networks needs the capability to detect and recognize this signalling.

A6. The function within the CSP's LI-system performing security processing (decryption), receives protected PDUs with the same or better quality than that of the PDUs arriving at the service endpoints (UE or STF).

EXAMPLE 4: The term "quality" could refer to PDU bit-errors, loss, or amount of reordering.

NOTE 2: A6 does not imply an assumption about reliable transport. It is fully within the scope to study handling of services transported over e.g. UDP.

A7. The LI product by the CSP needs to be complete and avoiding under-collection.

EXAMPLE 4: Start of intercept with already established session is to be supported (mid-session intercept).

A8. It is desirable that any proposed solution also works for services which are end-to-end protected between two UEs and using CSP-provided keys, as long as this is feasible without undue security or complexity impact.

NOTE 3: This is a consequence of A1. Assumption A1 makes it necessary for any solution to base its LI-functionality solely on the knowledge of the CSP-provided keys and in-band signalling traversing the CSP network.

A9. The UE and STF implementations are assumed to follow 3GPP specifications and have not been intentionally modified in order to bypass LI.

5 Key issues

5.1 Key issue: Cryptographic protocol considerations

5.1.1 Key issue #1.1: Security protocol detection

5.1.1.1 Key issue details

When using protocols such as AKMA, there is no explicit signalling occurring between the UE and STF before AKMA security can be taken into use. In other words, the first signs that CSP-provided keys might be taken into use is the start of a TLS (or some other) handshake. In particular in roaming, there is need for the VPLMN to be able to detect that an AKMA-based secure session is being started.

5.1.1.2 LI considerations

Since TLS and similar protocols can be used both with and without CSP-provided keys this creates the need to be able to distinguish sessions using CSP-provided keys from other sessions (e. g. using only server certificates). There can of course still be a desire to intercept also sessions not using CSP-provided keys, though such sessions can in most cases not be decrypted.

5.1.1.3 Potential LI requirements

A small set of well-defined protocol profiles are in use together with AKMA or similar key management service. These profiles are based on TLS or some other well-known protocol. The session handshake of these protocols comprises some easily detectable signalling that can tie the handshake to the use of CSP-provided keys. The (V)PLMN is equipped with logic that constantly monitors traffic between UE and potential STF, detecting session handshakes related to such protocols.

5.1.2 Key issue #1.2: Obtaining key management IRI

5.1.2.1 Key issue details

The most basic issue in order to allow the LI-system to decrypt traffic between a UE and the STF is that the LI-system gains access to the CSP-provided keys. In a non-roaming case, this can be done by leveraging the IRI-POI of the KSF, but there is no corresponding solution in the roaming case.

5.1.2.2 LI considerations

In roaming, it is undesirable for the VPLMN to request keys for LI-targets as this would reveal LI activation across PLMNs.

5.1.2.3 Potential LI requirements

In roaming, the HPLMN proactively pushes CSP-provided keys to the VPLMN, each time such a key is generated or updated on behalf of subscribers roaming into the VPLMN. Consequently, the key management functionality (KSF) of the HPLMN needs to have the ability to determine roaming status of subscribers. (In the case of AKMA, such functionality has been added, see TS 33.535 [11], clause 6.2.1.)

5.1.3 Key issue #1.3: Obtaining auxiliary security parameter IRI

5.1.3.1 Key issue details

Besides obtaining the keys as described in KI #1.2, there is also a need to obtain other information, needed to process the security protocol PDUs in the LI system. Information such as selected cryptographic algorithms, sequence numbers and nonces are typically also needed and such information is collectively referred to as auxiliary security parameters.

5.1.3.2 LI considerations

The auxiliary security parameters sought are in most cases only known to the STF and the UE and initial values for the parameters are exchanged in-band during security protocol handshake.

5.1.3.3 Potential LI requirements

The logic described in KI #1.1 to inspect security protocol handshakes is further equipped with functionality to extract relevant auxiliary security parameters from such signalling.

5.1.4 Key issue #1.4: Processing protected protocol PDUs

5.1.4.1 Key issue details

As discussed in clause 4.3, some decryption functionality of the LI-systems needs to mimic the behaviour of the decryption endpoint (UE or STF) in order to perform "security-decapsulation" of the security protocol PDUs, most importantly to decrypt them.

5.1.4.2 LI considerations

This is more challenging when the decryption functionality is not actively participating in the session, which is in turn necessary in order not to reveal that LI is activated.

5.1.4.3 Potential LI requirements

The (V)PLMN has processing logic that can mimic the processing (decryption) functionality of the receiving endpoint. To increase robustness of the LI-product, it is also desirable if this functionality verifies the data integrity of intercepted xCC.

5.1.5 Key issue #1.5: Processing encrypted handshake

5.1.5.1 Key issue details

The TLS 1.3 and DTLS 1.3 protocols allow for some of the handshake messages to be encrypted. These encrypted handshake messages can contain information which is needed in order to process/decrypt the remainder of the session. Examples of such information are handshake extensions (e.g. application protocol negotiation and auxiliary security parameters) and certificates (which could be useful for endpoint identification or UE-profiling).

5.1.5.2 LI considerations

Ability to decrypt the encrypted parts of the handshake could be necessary for ability to process/decrypt later application layer data and is generally useful for the resulting LI product.

5.1.5.3 Potential LI requirements

The processing logic of the (V)PLMN has the ability to also process/decrypt parts of the handshake that are encrypted.

5.1.6 Key issue #1.6: Processing early data

5.1.6.1 Key issue details

The TLS 1.3 and DTLS 1.3 protocols allow for application layer data comprising xCC to start to flow before the handshake is completed.

5.1.6.2 LI considerations

To avoid under-collection it is important to be able to process/decrypt also the early data.

5.1.6.3 Potential LI requirements

The processing logic of the (V)PLMN has the ability to also process/decrypt early data.

5.1.7 Key issue #1.7: Unreliable transport and PDU Fragmentation

5.1.7.1 Key issue details

DTLS messages, in particular during the handshake, can be larger than the standard 2^{16}-byte datagram size of UDP. For this reason, DTLS has a built-in fragmentation mechanism which allow DTLS messages to be fragmented. These messages need to be correctly reassembled before processing can start. The issue is more complex by the fact that the fragments can be encrypted and might be re-ordered during transmission.

5.1.7.2 LI considerations

Completeness of the LI product depends on the ability to re-assemble and process fragmented and out-of-order PDUs.

5.1.7.3 Potential LI requirements

Buffering is provided within the LI-system and is used for reassembly of PDUs before other cryptographic processing the PDUs.

5.1.8 Key issue #1.8: Perfect forward secrecy

5.1.8.1 Key issue details

Perfect Forward Secrecy (PFS) is built into many protocols such as TLS or DTLS. The UE and STF runs a so-called Diffie-Hellman exchange which produces a second key, which is combined together with the CSP-provided key material, forming the complete session main key. This Diffie-Hellman key is usually known only to the UE and the STF and can thus not be obtained as xIRI from the LI-system. This holds regardless of whether the KSF is located at the VPLMN or the HPLMN.

5.1.8.2 LI considerations

Cipher suites using PFS cannot be decrypted even if they are using CSP-provided keys.

5.1.8.3 Potential LI requirements

The cipher suites used with CSP-provided keys need to be configurable to not make use of PFS, or, to use PFS based on keys provided by the CSP.

5.1.9 Key issue #1.9: Session resumption

5.1.9.1 Key issue details

Protocols such as TLS have a built-in session resumption mechanism which allows the handshake on sub-sequent handshakes to be shortened. The UE and STF will store a resumption secret from a previous session and indicates desire to reuse this secret as key, by sending a so-called ticket, pointing to the previous secret.

5.1.9.2 LI considerations

Obviously, if LI was not activated already on the previous session which created the resumption secret, it will not be possible to intercept the sub-sequent session either.

5.1.9.3 Potential LI requirements

The protocol profiles used between UE and STF are configured to not use resumption or to issue resumption information such as tickets for potential future use.

5.2 Key issue: Mid-session intercept

5.2.1 Key issue #2.1: Handshake dependency

5.2.1.1 Key issue details

As already discussed, the handshake initiating a session contains important information (auxiliary security parameters) which is necessary to derive the session keys. This creates an issue in activating LI for a session that has already been started when a warrant is received.

5.2.1.2 LI considerations

To support mid-session activation of interception, the LI-system needs access to information from a handshake that took place before LI was activated.

5.2.1.3 Potential LI requirements

The LI system is equipped with a function that extracts and stores auxiliary security parameters from security handshakes even when LI is not yet activated.

5.2.2 Key issue #2.2: Obtaining cryptographic context synchronization

5.2.2.1 Key issue details

As described in clause 4.3, the state information that is needed to process a security protocol PDU is updated for each new PDU. In a typical case, it might be necessary to know the value of a packet counter in order to carry out security processing. The correct value of this information depends not only on auxiliary parameters exchanged during the handshake as described in KI #2.1, but is further updated for each new security protocol PDU.

5.2.2.2 LI considerations

If LI was not activated from the start of the session, difficulty in knowing the value of synchronization information can arise.

5.2.2.3 Potential LI requirements

The LI system is equipped with a function that extracts and stores synchronization information from security handshakes even when LI is not yet activated. Further, this function keeps the synchronization information up-to-date, in preparation for future activation of LI.

5.3 Key issue: Roaming and trust model

5.3.1 Key issue #3.1: Inter-PLMN dependency

5.3.1.1 Key issue details

In a roaming scenario, the KSF is located in the HPLMN. The key material for a potential LI target might not be known in the VPLMN, unless the STF also resided in therein.

5.3.1.2 LI considerations

The VPLMN cannot actively request keys for LI targets (as this would reveal LI activation to the HPLMN), there is therefore a need for the HPLMN to push keys to the VPLMN each time a key is created or updated.

5.3.1.3 Potential LI requirements

The HPLMN needs to push keys to the VPLMN each time a key is created or updated.

5.3.2 Key issue #3.2: Separation of data confidentiality and integrity

5.3.2.1 Key issue details

Most security protocols have provisions for both data integrity and data confidentiality. It is also common that the keys for both security functions are derived from the same base key (or are even identical). In the case of AKMA being used together with TLS for example, both keys are derived from a common K_{AF} key.

5.3.2.2 LI considerations

For LI-purposes, the simplest approach is that the LI-system (or the LEA) is given access to a common base-key which enables both decryption and data integrity verification. On the other hand, this also opens the possibility that a compromised or malfunctioning LI-function could produce secured PDUs which cannot be distinguished from PDUs produced by the LI-target UE. While the idea that this could happen might seem far-fetched, it appears to always be strictly preferred if integrity could be maintained end-to-end while still enabling decryption of the LI-product.

Solutions to this key issue could in part be non LI-specific but rather have implications for the details of the key management of the security protocol used and/or the solution for how the CSP provides the keys.

5.3.2.3 Potential LI requirements

It is desirable to look at solutions which allow the LI-system (and/or LEA) to only decrypt target traffic but without having any capability to affect the data integrity. As a special case of this, one could enable solutions that only provide integrity protection, without activating encryption.

5.4 Key issue: LI architecture

5.4.1 Key issue #4.1: Decryption functionality

5.4.1.1 Key issue details

The ultimate goal of the present document is to present solutions which enables the LEA to obtain access to plaintext communication for a target who uses a service encrypted by CSP-provided keys. This can be achieved in two main ways characterized by either:

1. The CSP providing LEA with encrypted CC and IRI comprising decryption keys and other information enabling decryption to be performed at the LEA.
2. The CSP using decryption keys and other information, decrypting xCC and then delivering plaintext CC to the LEA.

5.4.1.2 LI considerations

In the present document, only option 2 is studied. This is motivated as follows:

- It is believed that most LEAs would prefer to obtain already decrypted CC, avoiding the need for additional processing at the LEA.
- Mid-session start of intercept is in any case judged necessary to handle internally to the CSP since otherwise, as discussed in KI #2.1 and #2.2, information also for non-targets would need to be made available to the LEA.
- Option 2 removes the need to expose key material outside the CSP (though a secure handover interface is of course still needed).
- Option 2 is more technology-neutral for the LEA, since it limits the need for LEA equipment to be upgraded as security protocols are updated or new ones emerge.
- Given the necessary information (keys and other parameters) the technical details on how to implement the decryption functionality is largely independent of which of the two options that is chosen. By studying case 2 only, no relevant analysis details are lost that would prevent a deployment of option 1, should that still be preferred.

Therefore, the main consideration for the present document is to analyse a suitable architectural location of the decryption functionality within the CSP and to investigate how this functionality could operate.

5.4.1.3 Potential LI requirements

The LI architecture needs to consider and propose suitable architectural location(s) of the decryption functionality and its interactions with other components of the LI-system.

5.4.2 Key issue #4.2: Provisioning and triggering

5.4.2.1 Key issue details

The cases considered in the present document are:

- A non-roaming case with STF located outside the CSP (at an external DN), and,
- The roaming case with the STF located outside the VPLMN CSP (at an external DN or in the HPLMN).

A common property of both cases is that provisioning of the STF is not feasible (assumption A1). Additionally, with key management solutions such as AKMA, there is no possibility to a priori identify *which* STF that might take the key material into use in communicating with a target UE, nor *when* this key material might be taken into use.

5.4.2.2 LI considerations

As discussed above, there will inevitably be a need to base the LI solution on triggering, at least of the functionality that performs the actual xCC decryption (i.e. over LI_T3 or a special triggering interface for encrypted services).

It might still be possible to use provisioning to some extent. This could apply for example in the roaming case: some LI function in the VPLMN will likely be given the role of receiving key material for inbound roamers from the HPLMN, and this LI function could be provisioned over LI_X1.

5.4.2.3 Potential LI requirements

The LI solution needs to define procedures for how and which LI-functions to provision and trigger.

5.5 Key issue: Security assurance

5.5.1 Key issue #5.1: Roaming interface security

5.5.1.1 Key issue details

The roaming interfaces already today carry cryptographic key material (authentication vectors) implying high security requirements for the roaming interfaces (N32), e.g. applying PRINS of TS 33.501 [10] or similar solutions. However, the keys that are currently transported over these interfaces are typically used to protect the LTE/NR radio access, the NAS protocol, or AKMA services between the UE and an STF located in the VPLMN. In other words, the keys currently transported are *intended* for usage between the UE and the VPLMN. To allow encryption to be used *across* the VPLMN while not causing conflict with the VPLMN's LI-obligations, keys need to be made available to a VPLMN that would otherwise not have access to them.

5.5.1.2 LI considerations

Transporting new types of encryption keys over the roaming interface could introduce additional risks that need to be mitigated. However, since the keys (as discussed in KI #1.2) need to be pushed from the HPLMN to the VPLMN regardless of LI, the considerations are not LI-specific but is rather a general roaming interface security issue.

5.5.1.3 Potential LI requirements

There are no additional requirements on the LI-system itself, and instead a review of roaming interface security seems appropriate in order to evaluate if modifications to PRINS or other solutions are needed.

5.5.2 Key issue #5.2: Decryption functionality security

5.5.2.1 Key issue details

The LI-system will be given the ability to access subscriber traffic that is encrypted everywhere outside the LI-system.

5.5.2.2 LI considerations

It is crucial that the LI-functions are implemented with a security assurance level so that risks related to exposure of data which is otherwise encrypted across the entire CSP network is mitigated. Currently, the security requirements on the LI-system are that its output is not accessible by any unauthorized party outside the LI-system itself.

5.5.2.3 Potential LI requirements

While the current security requirements on the LI-system appear qualitatively sufficient, more detailed and specific security assurance requirements for the decryption functionality within the LI-system ought to be studied to ensure this level is maintained.

5.6 Key issue: UE-to-UE protected communication

5.6.1 Key issue #6.1: Interception of UE-to-UE E2E protected communication

5.6.1.1 Key issue details

As long as the keys are CSP-provided, there seems to be no new technical elements introduced if one considers UE-to-UE encrypted communication, rather that UE-to-STF.

NOTE: It is not likely that protocols such as AMKA would be used in this case, a more probable solution would be a CSP-operated key management server or similar, as described in the "ticket-based solution" of TR 33.828 [12].

5.6.1.2 LI considerations

Though no new technical elements enter, there seems however to be trust model related issues: there could now be four different entities involved: the HPLMN and VPLMN of the first UE, as well as the HPLMN and VPLMN of the second UE. These four could be associated with different CSPs in different jurisdictions.

5.6.1.3 Potential LI requirements

Feasibility of LI for services encrypted by CSP-provided keys ought to be analysed, considering different use-cases related to the number of jurisdictions and CSPs involved, and relevant trust and business models involving these parties.

6 Solutions

6.1 Mapping of solutions to key issues

Table 6.1-1: Mapping of solutions to key issues

Solution	Key Issue																	
	1.1	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9**	**2.1**	**2.2**	**3.1**	**3.2**	**4.1**	**4.2**	**5.1**	**5.2**	**6.1**
1.1	X																	
1.2		X																
1.3			X															
1.4				X	X	X	X											
1.5								X	X									
2.1										X	X	X						
3.1												X			X	X		
3.2													X			X	X	
4.1										X	X				X			
4.2														X	X			
5.1																	X	
6.1																		X

6.2 Solutions for cryptographic protocol considerations

6.2.1 Solution #1.1: Security Protocol Detection Function (SPDF)

6.2.1.1 Introduction

The CSP needs means to detect that:

a) The UE is initiating use of a security protocol with some STF, and,

b) That the secure protocol will be using keys provided by a CSP.

In roaming, the two CSPs mentioned above can be two separate CSPs. Regarding (a), all security protocols that will realistically be in use, always comprise an initial handshake, as discussed in clause 4.3.1. Point (b) is specific to details of the solution used for CSP-provided keys, but it is reasonable to expect that some key identifier will always be included in the handshake, see solution #1.2 below for a specific case.

6.2.1.2 Solution details

The functionality needed resembles that of the BBIFF used in N9HR. Just as the BBIFF notifies LI-functions (specifically, the LMISF) when it detects that a UE establishes IMS-related bearers for signalling or media, the purpose of the security protocol detection function is to detect when a UE establishes a secured (encrypted) session with some service, and where the encryption is based on CSP-provided keys. Some suitable NF of the CSP inspects all UP traffic of the subscriber, looking for security protocol handshake initiations by the UE. A first level selector would be to look at port usage since many security protocols use well-known ports. When a security handshake is detected, the function further inspects the handshake more deeply, looking for usage signs of CSP-provided keys, e.g. key identifiers.

For performance reasons, this function, henceforth denoted Security Protocol Detection Function (SPDF), would suitably be co-located with some NF that is always present in the UP-path in the CSP network, e.g. the UPF (in roaming: vUPF). Figure 6.2.3.2-1 (below in clause 6.2.3.2) shows the SPDF residing with a UPF CC-POI as an example.

Observe that there could be cases where LI-activation has already been performed when a security handshake occurs, in which case the SPDF can be assumed to already be configured with target information. However, there can also be cases where the security handshake occurs directly in conjunction with PDU session establishment, in which case the SPDF would need to be triggered, e.g. as part of LI_T3 triggering of LI functions of the UPF. If mid-session intercept is to be supported, there can also be cases where not even triggering can be assumed to take place, see clause 6.3.1.

6.2.1.3 Evaluation

The SPDF solution outlined is judged to be technically reasonably simple to implement, though some processing capacity requirements are implied, see clause 6.2.3.3. The exact details will depend on details of the solution for CSP-provided keys and which security protocol(s) that might be used with these keys. As the SPDF is most likely tightly integrated with the UP path, the functionality can be fully standardised within 3GPP SA3-LI and adoption is limited to CSPs using 3GPP standards.

6.2.2 Solution #1.2: Use of AKMA (or equivalent)

6.2.2.1 Introduction

This solution shows an instantiation of solution #1.1 for the concrete case that AKMA is used and illustrates in more detail how key management IRI are extracted from UE-STF signalling. Usage of AKMA here includes using one of the pre-specified, TLS-based Ua* protocols defined in TS 33.222 [9].

6.2.2.2 Solution details

TLS handshakes most commonly use port 443 for HTTPS-based services (though other ports might be defined for other services) and is therefore a suitable first level selector by the SPDF. The first messages of the actual TLS handshake is always a ClientHello and is not encrypted (see the evaluation below). The initial response back from the server (ServerHello) is also (at least partly) unencrypted. This therefore provides reliable means to detect usage of TLS.

To determine IRIs related to key management, it is necessary to handle TLS 1.2 and 1.3 separately.

In TLS 1.2, when used with AKMA, the (unencrypted) TLS handshake messages ServerKeyExchange and ClientKeyExchange can be used as a verification of the use of AKMA key material as follows. The PSK-identity field of the ServerKeyExchange handshake message will have the value "3GPP-AKMA" and the corresponding field of the ClientKeyExchange message will have the value "3GPP-AKMA" followed by the AKMA A-KID if AKMA is taken into use. Further, the server_name extension to the TLS 1.2 ClientHello could also be useful since this indicates the

AKMA AF hostname. This is useful if the provided K_{LI} decryption key is the AKMA anchor key (K_{AKMA}) rather than the AF-specific key K_{AF}, since derivation of K_{AF} (depending on the AF host name) is then necessary.

If TLS 1.3 is used, the UE indicates use of AKMA, not in a ClientKeyExchange handshake message, but rather as part of a PresharedKey extension to the ClientHello. The format is however the same, the PSK identity field is set to "3GPP-AKMA" followed by the A-KID.

6.2.2.3 Evaluation

The described solution appears feasible as a basis for interception of AKMA-based TLS services. The existing LI-solution for AKMA can be reused (or extended within 3GPP SA3-LI responsibility, if required). There are however some caveats regarding the effectiveness which will be discussed as part of solution #1.5.

There is some dependency on ongoing work outside 3GPP. IETF is working to define an encrypted version of the ClientHello, using a public key associated with the server. It is not yet clear if and how 3GPP will adopt this, and this could significantly impact the feasibility of this solution.

6.2.3 Solution #1.3: Security Handshake Interception and Forwarding Function (SHIFF)

6.2.3.1 Introduction

Besides the cryptographic keys which are obtained as xIRI from the KSF, auxiliary security parameters are also needed as described in KI #1.3. Under assumption A.1, this is only available as UP in-band information exchanged between UE and STF during a handshake.

6.2.3.2 Solution details

The procession logic of the SPDF proposed in solution #1.1 for security protocol detection can be extended with more full-fledged POI-functionality that, after security protocol detection, proceeds to intercept the full handshake. Optionally, the handshake interception functionality is a separate entity, the Security Handshake Interception and Forwarding Function (SHIFF). Technically, the intercepted information can be viewed as IRI since it comprises metadata related to the security protocol, though the interception is architecturally done in the UP between UE and STF, e.g. at a POI co-located with the (v)UPF. Figure 6.2.3.2-1 shows both the SHIFF and the SPDF of solution #1.1 as separate functional entities co-located with a UPF CC-POI.

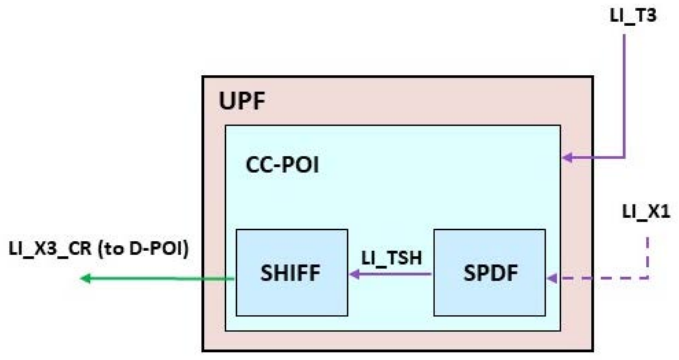

Figure 6.2.3.2-1: SHIFF and SPDF within a UPF CC-POI.

Both functions could most likely reuse the LI_T3 interface of the CC-POI, though provisioning might also apply in some cases. This could happen as follows:

1. A PDU session is established by the SMF, which results in triggering the CC-POI of the UPF over LI_T3.

2. The PDU session proceeds for a while without any security protocol taken into use. If there are keys available at the CSP (for example, at the SEAF, as described in solution #3.1) the SPDF could at this point be provisioned by these keys over LI_X1, in preparation for a possible security handshake.

3. Later, when the security handshake is detected at the SPDF, the keys can be taken directly into use to decrypt traffic.

An interface LI_TSH (LI Triggering Interface for Security Handshake) is proposed to be used by the SPDF when it detects a security protocol handshake relevant for interception, i.e. one that uses CSP-provided keys. Usage of this interface would be relevant mainly to trigger the SHIFF into collecting auxiliary security parameters in preparation for a potential later mid-session intercept (see solution #2.1). If LI is already activated, interception would instead start immediately. Further, an interface LI_X3_CR is proposed to be used to forward auxiliary security parameters, extracted from the handshake, to the entity performing the actual decryption (the Decryption POI, D-POI, discussed in clause 6.5.2). The suffix "X3_CR" is chosen to signify that the interface is used to transfer crypto-related parameters captured from UP (xCC) traffic. For mid-session intercept, the receiving entity of the intercepted handshake information would store it in preparation for a possible later LI-activation.

NOTE: Since co-location of the D-POI with the CC-POI of the UPF appears an advantageous architectural solution (see analysis of clause 6.5.2), the LI_X3_CR interface would in this case be internal to the CC-POI. An external LI_X3_CR interface would however still be relevant to support mid-session intercept, see discussion in clause 6.5.1.

Provisioning/configuration of SPDF over LI_X1 could also be relevant when the SPDF is used to enable mid-session intercept, see the discussion on solution #4.1 in clause 6.5.1.

There is an open issue on the location of the SHIFF relative to D-POI. If mid-session intercept is to be supported, the SHIFF would typically need to be invoked independently of the D-POI, whereas if it is not to be supported, or, if the security protocol uses an encrypted handshake, the SHIFF would benefit from tighter integration with the D-POI, see further discussion in clause 7.5 and 8.

6.2.3.3 Evaluation

Technically, the solution would bootstrap on (e.g. be triggered by) solution #1.1 and can be handled entirely within 3GPP SA3-LI defined standards. While the basic technical principles are straightforward, it needs to be noted that the inspection done in solution #1.1 and the following interception performed as part of this solution needs to be done at line-speed. A technical consideration that needs handling is that there is in general no one-to-one correspondence between security protocol PDUs and transport layer datagrams. In fact, there can even be fragmentation within the security protocol itself. For example, large (D)TLS Handshake messages can be split across several (D)TLS records (and thus also over several TCP/UDP packets). Conversely, a single UDP/TCP-packet can contain several (D)TLS messages.

6.2.4 Solution #1.4: Security processing state machine

6.2.4.1 Introduction

In order to deliver an LI-product consisting of plaintext decrypted target UP traffic, the LI-component responsible for this needs to mimic the complete state machine of the receiving end-point as described in KI #1.4, #1.5 and #1.6, and possibly also cope with unreliable transport and fragmentation (KI #1.7). This contrast somewhat with current POI-type functions, that mainly just need to capture the right packets.

6.2.4.2 Solution details

The responsible LI-component (e.g. a CC-POI) implements the "full-stack" state machine of that (or those) security protocol(s) that are within the scope of interception. With reference to the high-level overview of how such a state machine would operate in clause 4.3, the following process would take place in the LI-system. The steps serve similar purposes as those of the actual protocol endpoints as discussed in clause 5.1, though the relevant information needed here is partly based on provided xIRI since the LI-system is not actively participating in the protocol.

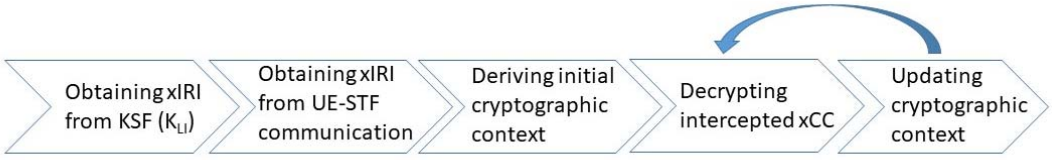

Figure 6.2.4.2-1: LI state machine overview, with LI activation prior to encrypted session start. The step "Decrypting intercepted xCC" follows the outline presented in clause 4.3.3.

More concrete examples in the case of TLS/DTLS are provided in annex A.

6.2.4.3 Evaluation

The solution adds complexity to the LI-component corresponding to that of a full server implementation of the respective security protocol(s) which might seem substantial. However, there is no shortage of open-source distribution for all interesting security protocols and most LI-components would anyway need to implement TLS or a similar protocol in order to secure connections to/from ADMF, MDF, or other LI-components. There are some new elements that enter into the picture when the decryption functionality is not actively participating as an endpoint, but those elements become noticeable mainly in conjunction with support for mid-session intercept (see solution #2.1 and annex A).

There is a performance issue to consider since the decryption processing overhead grows linearly with the number of UP sessions being intercepted, whereas the overhead of normal LI usage of TLS only grows with the number of internal LI-interfaces.

The solution is purely up to the implementation of 3GPP SA3-LI-defined functionality for CSP usage and does not require additional standardisation outside 3GPP.

6.2.5 Solution #1.5: Cipher suite profiling

6.2.5.1 Introduction

Assuming the security protocol detection and key management IRI extraction as discussed in solution #1.1 and #1.2 are in place, there remains some issues related to the actual cipher suite in use as described by key issue #1.8.

First, solution #1.2 is only effective if the authentication schemes used are one of those defined in clause 5.4 of TS 33.222 [9] that only relies on AKMA-keys as pre-shared keys, and without use of certificates (clauses 5.3 and 5.5 of TS 33.222 [9]). If one of the latter approaches are used, although the handshake might still be detected and relevant IRI might be extracted, that will not enable interception of the service traffic that follows the handshake, since that will not be encrypted based on the AKMA keys. Secondly, even if the approach of clause 5.4 of TS 33.222 [9] is used, there remains issues related to which cipher suite is taken into use since some of the cipher suites of the Ua* protocol profiles defined by TS 33.222 [9] combine the AKMA keys with Diffie-Hellman keys that are known only at the UE and STF.

Finally, protocols such as TLS allows issuing "tickets" as discussed in relation to KI #1.9 which would make it infeasible to perform interception of sessions that are resumed from sessions that occurred prior to LI activation. Similar problems arise with protocols that "refresh" keys during a session, see clause A.4.3.4.1 for discussion.

6.2.5.2 Solution details

The proposed solution is more of a policy solution than a technical one. The solution consists in configuration of the Ua* protocol usage (in UE and STF) to only make use of the AKMA authentication approach of TS 33.222 [9] clause 5.4, and to only make use of Ua* protocol cipher suites that rely solely on the CSP-provide keys, without usage of PFS mechanisms.

6.2.5.3 Evaluation

The solution is technically straightforward and effective under assumption A9.

The current cipher suites specified in 3GPP are not compliant with the above solution and would need new specification work by 3GPP SA3. The new cipher suites would need to be adopted in UE implementations.

6.3 Solutions for mid-session intercept

6.3.1 Solution #2.1: Security state mirroring

6.3.1.1 Introduction

If a warrant is to be activated on a secure protocol session that has already started some time ago, it is necessary to retrieve information from the previous security protocol handshake according to KI #2.1 and it is also necessary to "fast-forward" the security processing state from the handshake to the security protocol PDUs currently being forwarded as discussed in KI #2.2.

6.3.1.2 Solution details

To support mid-session intercept, it is proposed that the solutions #1.1 and #1.3 for handshake detection and interception (the SPDF and SHIFF) are generalized and extended by:

1. A preparatory handshake capture function that captures handshake messages (even for sessions that are currently not being intercepted), and,

2. A basic keep-in-synch mechanism thar records and stores information related updates of the cryptographic state at the actual endpoints.

In a simple case, the function (1) would just need to capture and store relevant auxiliary security parameter IRI, extracted from the handshake. For more complex protocols (where handshake messages might be encrypted) raw PDU-captures might need to be stored, unless the decryption keys are already available.

The function (2) would always comprise a function that counts security protocol PDUs, so that the correct sequence number is known when a warrant is received. In some cases (depending on the protocol) the function (2) would also capture specific security protocol messages that are used to modify keys during the session. If keys to process these are available, the keys can be updated in real-time, whereas if the keys are not yet available, raw copies might need to be stored until keys do become available.

Figure 6.3.1.2-1 below illustrates how the "security state mirroring" just described would fit with, and support, the LI process. During state mirroring, necessary IRI from UE-STF communication (security protocol handshake) is obtained by a concurrently running security-state mirroring function which is configured to recognize security handshakes of relevant security protocols (as outlined in solution #1.1) and to extract relevant IRI from these handshakes in case they are later needed to start interception of sessions that has already started. This mirroring occurs independently of identities of currently known LI-targets.

The extent to which the process can actually extract individual IRI depends on details of the security protocol, e.g. whether handshake messages are encrypted and whether they can even be recognized as handshake messages without the ability to decrypt as will be elaborated in more detail in annex A, for specific security protocols. When this is not possible, the state mirroring has no option but to capture more or less all of the initial security protocol PDUs.

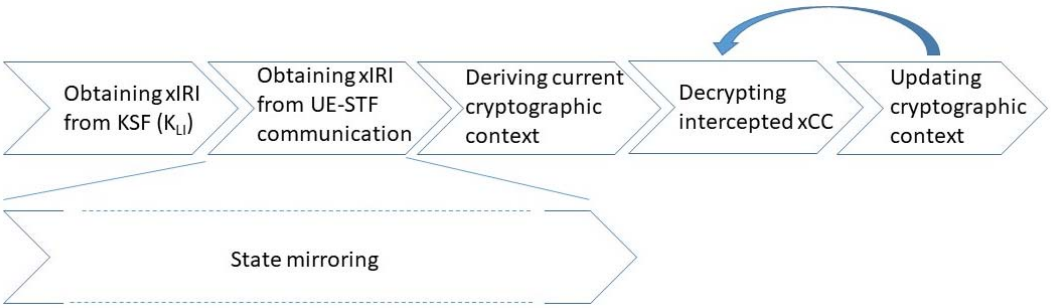

Figure 6.3.1.2-1: LI process overview for LI activation with established encrypted session security using state mirroring. The step "Decrypting intercepted xCC" follows the outline presented in clause 4.3.3 and starts if/when LI is activated.

6.3.1.3 Evaluation

The main part of the solution is a storage function and a reasonably simple control/synchronization mechanism. The main complexity is the amount of storage needed. A typical TLS handshake consists of about half a dozen messages, each limited in size by 2^{16} bytes (in practice, usually much smaller). A complete TLS handshake can therefore typically be assumed to require a few hundred kB of storage. This assumes raw storage is needed, i.e. that the keys to process the handshake are not yet available, otherwise the storage can probably be made significantly smaller. Besides the storage requirement, there is also an implied, and probably more challenging requirement that the state mirroring needs to be performed at line speed.

The solution needs to be able to handle security protocol PDUs that are split across packets/datagrams as discussed in relation to solution #1.3. Whether assembly of fragments into complete security protocol PDUs are possible to be done in an integrated fashion depends on specifics of the protocol, e.g. whether the individual fragments are encrypted or not. When not possible, re-assembly needs to be deferred until LI activation (when the decryption keys become available).

In roaming, to have a workable LI-solution it would in most cases be necessary that the keys *are* indeed already available in the VPLMN in order to avoid requesting keys for LI-targets from the HPLMN. Therefore, ability of complete processing of a (possibly) encrypted handshake would in most cases already be available as the handshake occurs, thereby limiting storage needs and greatly simplifying a later LI-activation. Indeed, needing to decrypt and process the handshake when LI is activated would incur delay in LI-activation and might pose a risk of race conditions or under-collection.

However, even if the keys are available as the handshake occurs, there could be some security advantage in keeping access to the keys separated from the storage function until a warrant is actually received, so this will come with a security/complexity trade-off.

The functionality lies entirely within 3GPP SA3-LI mandate, assuming the LI-system has access to the relevant keys.

6.4 Solutions for roaming

6.4.1 Solution #3.1: Extended KSF and SEAF functionality

6.4.1.1 Introduction

The keys for the secure UE-STF connection are (for key management solutions similar to AKMA) created by the KSF on request by the STF, which in turn occurs as the UE sets up a secure connection with the STF. In roaming, to support LI in VPLMN the keys need to be made available to the VPLMN in a secure manner. Further, the keys need to be provided from the HPLMN without the HPLMN knowing if or when the corresponding subscriber would be an LI target.

6.4.1.2 Solution details

A natural solution is that the KSF actively pushes keys to some NF in the VPLMN each time a key is produced. As discussed, the AKMA-solution has functionality that enables the AKMA KSF (the AAnF) to determine the roaming

status of the UE. Thus, for AKMA, this could be done by the AAnF in conjunction to producing the AKMA anchor key (K_{AKMA}), or each time an STF (i.e. an AKMA AF) requests a key (K_{AF}). In any case, this needs to be done over a non-LI interface, which for the purpose of the present document is assigned the working name "Nk".

In the 5G context, a suitable location to receive the key-push would be the SEAF, since its role is that of a security anchor in the VPLMN, receiving and maintaining various keys for a UE. The solution would then consist of a new inter-PLMN reference point between the KSF (AAnF) and the SEAF.

NOTE: The SEAF functionality is in the current 5G architecture assumed co-located and hosted in the AMF.

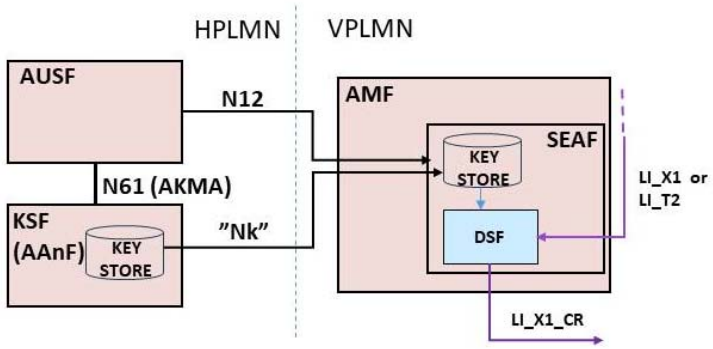

Figure 6.4.1.2-1: Possible SEAF-internal functional entities. Keys (e.g. AKMA keys) are here assumed pushed over the "Nk" interface which could be secured by the N32/PRINS solution (not shown). Observe that "Nk" is not an LI-interface, but part of the network architecture for roaming.

In the current 5G architecture, the SEAF receives a key denoted K_{SEAF} from the AUSF over the N12 reference point, K_{SEAF} being produced as result of primary authentication. This could be extended by also having the AAnF push AKMA keys to the SEAF as result of key requests by AKMA AFs. The illustration below shows a potential integration with 5G SBA.

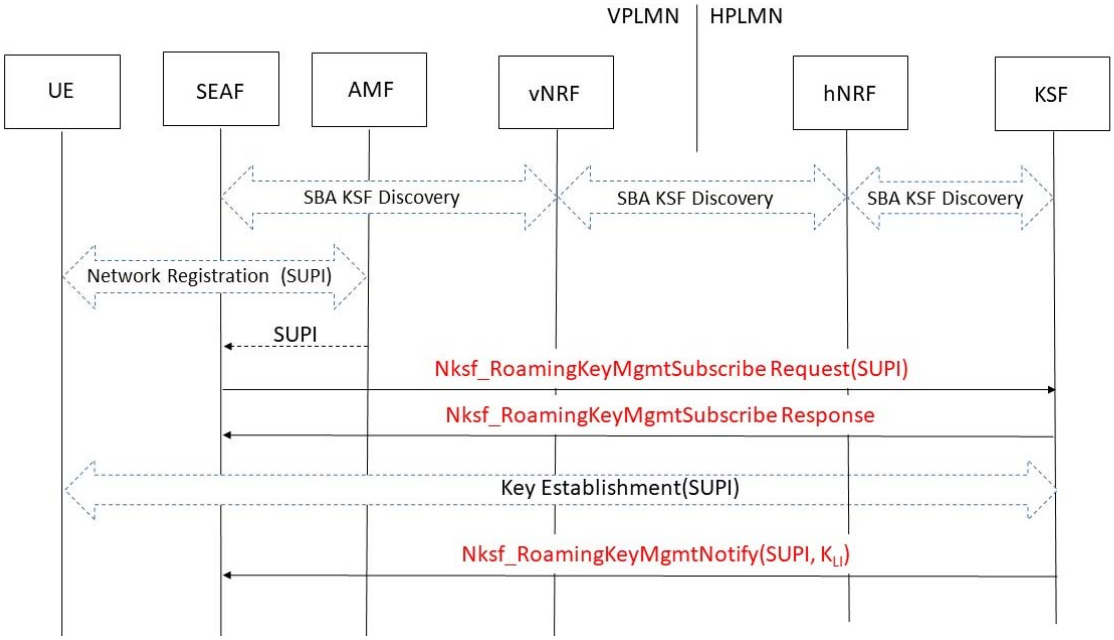

Figure 6.4.1.2-2: Possible SBA-integration with new SBA services highlighted in red. While SEAF is currently co-located with AMF, it is here shown separately for clarity (SEPPs are not shown).

The principle would be that the VPLMN SEAF first subscribes to key management notification (e.g. creation of new keys) from the HPLMN KSF for each SUPI corresponding to an inbound roamer. Subsequently, the KSF notifies the SEAF of corresponding events, including the SUPI and the key(s) relevant to perform LI. It is proposed that a Decryption Support Function (DSF) is provided in the SEAF to handle keys received from the HPLMN.

Once received by the DSF, a copy of the pushed key(s) would be accessible to an IRI-POI associated with the SEAF which, depending on the use-case could be provisioned or triggered. From there on, the key would be transferred over an interface denoted LI_X1_CR to VPLMN POIs needing them to decrypt traffic. The notation "X1_CR" is used to signify an interface provisioning crypto-related parameters.

6.4.1.3 Evaluation

The SEAF is currently collocated with the AMF and there is thus no dedicated POI for the SEAF.

For AKMA, the product implementation impact falls on the AAnF and the SEAF since these NFs needs to implement the new Nk interface.

Since the SEAF is from the outset designed to handle cryptographic keys, there seems to be no additional assurance requirements beyond what already exists.

The DSF functionality of the SEAF and its LI-interfaces are fully within 3GPP SA3 LI standardisation mandate. Defining an Nk reference point with suitable security measures would fall on the responsibility of 3GPP SA2/SA3. However, it appears feasible to reuse the N32-security solution to protect also Nk without need for additional specification work.

6.4.2 Solution #3.2: Key hierarchy and key separation

6.4.2.1 Introduction

As described in KI #3.2 it would be desirable if the ability for the LI-system to decrypt still limits the risk that a malfunctioning or incorrectly implemented (KI #5.2) LI-system could also end up modifying or generating CC that appears to come from the LI-target. This would also reduce threats relevant for the roaming case (KI #5.1). The problem is that solutions such as AKMA has an "all-or-nothing" type of key-management: the generated key enables both decryption and data authentication/integrity. A single key is provided by the KSF (AAnF) to the STF (AF), and this key is then taken into use with TLS or similar protocol. Internally, this protocol might generate separate encryption and integrity keys, but as long as the LI-system can only obtain keys as IRI from the KSF, this means that the LI-system will also have access to the same key as the STF, and therefore also have access to all keys used by the security protocol.

6.4.2.2 Solution details

The solution is best described by an example using AKMA. Rather than generating a single key for the AKMA AF:

K_{AF} = KDF(K_{AKMA}, AF-identity, …);

where KDF is a key-derivation function and AF-identity is an identifier for the AF, the AAnF instead generates two keys:

$K_{AF, AUTH}$ = KDF(K_{AKMA}, AF-identity, "AUTH", …), and,

$K_{AF, ENC}$ = KDF(K_{AKMA}, AF-identity, "ENC", …),

to be used for authentication and encryption purposes, respectively.

The (only) key provided to the LI-system (possibly by first sending it over the roaming interface Nk as discussed in solution #3.1) is $K_{LI} = K_{AF, ENC}$., whereas both keys are provided to the STF (AF).

Most current protocols such as TLS use a single "master key" to generate both encryption and authentication/integrity keys, so the solution would need to be complemented by new cipher suites that separates key usage and allows to separate master keys to be used.

6.4.2.3 Evaluation

The solution is technically straight forward and would achieve the goals. However, while the possibility to extend the AKMA key derivation functionality falls entirely within 3GPP SA3 mandate, it would be beneficial if the new cipher suites could also be standardized outside 3GPP, e.g. IETF. It is not clear if enough interest exists in IETF to take on that task.

By arranging so that the authentication/integrity key does not need to be sent outside the HPLMN, it further reduces risks related to roaming interface security.

A potential issue is that a UE that colludes with the STF could bypass LI by simply using $K_{AF, AUTH}$ also for encryption, but that would fall outside the scope according to assumption A9.

6.5 Solutions for LI architecture

6.5.1 Solution #4.1: LI Mirror Security State Function (LMSSF)

6.5.1.1 Introduction

To support a mid-session intercept solution that can address KI #2.1 and #2.2, an approach to state-mirroring was proposed in solution #2.1. The functionality of solution #2.1 needs to be hosted and provisioned within the LI-architecture and this naturally leads to defining an LI Mirror Security State Function (LMSSF) as outlined in the present clause.

6.5.1.2 Solution details

Solution #1.1 and #1.3 proposes new LI functionality that monitors security handshakes and extracts information from them, enabling intercept to be started at a later point in time. The information is managed by a security state mirroring process as laid out in solution #2.1. Further, common to all of these functionalities are that they need access to UP traffic only. As already described, it is natural to consider co-locating solution #1.1 and #1.3 in a UP entity, which extracts and forwards information to the security state mirroring. The latter function shows many similarities with the LMISF used for S8HR/N9HR so it natural then to define a LI Mirror Security State Function (LMSSF). From this function, necessary information (IRI) could be retrieved when LI is triggered or provisioned, so it also forms a basis for addressing KI #4.2.

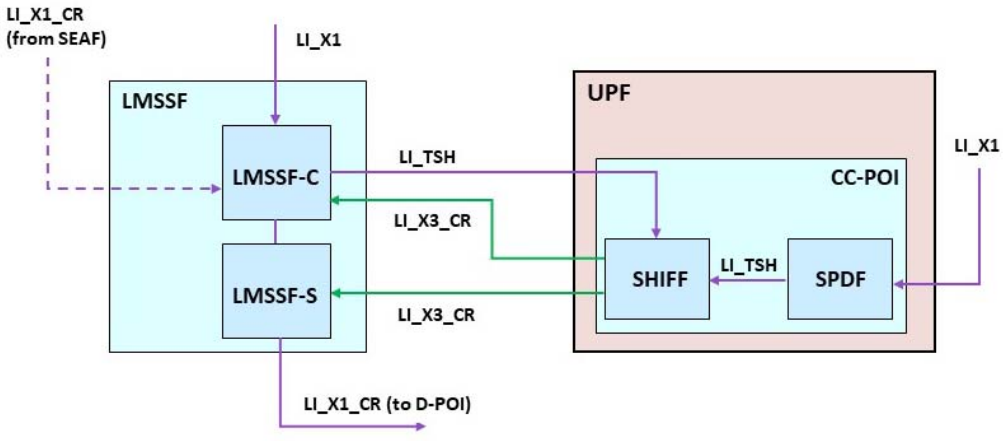

Figure 6.5.1.2-1: LMSSF within the LI-architecture. The LI_X1_CR interface from SEAF is optional, see discussion below.

The LMSSF comprises a control part (LMSSF-C) and a storage part (LMSSF-S). The principles of operation would be as follows:

- To enable mid-session intercept, the LMSSF and the SPDF are first provisioned over LI_X1 to monitor for relevant security handshakes. This provisioning is typically not target specific.

- When the SPDF detects such a handshake, it triggers the SHIFF over LI_TSH to start capturing the handshake, and to notify the LMSSF-C that it has detected the handshake.
- The SHIFF notifies LMSSF-C over LI_X3_CR.
- The LMSSF-C takes a decision whether the handshake is to be captured to enable a potential later mid-session activation of LI.
- If the LMSSF-C decides that capture it is performed, it sends a confirmation trigger to SHIFF over LI_TSH. This message also instructs the SHIFF about the need to keep monitoring the security protocol progress in order to also capture and forward messages after the handshake, the content of which might affect the current security state.

NOTE: As previously discussed, key update messages, security PDU counters, etc, are examples of information that might need to be captured after the initial handshake.

- The SHIFF forwards the handshake (and potentially later occurring security state related information) to the LMSSF-S over LI_X3_CR.
- If, at a later time, a warrant to activate LI is received, the LMSSF is provisioned (in a target-specific manner) over LI_X1, and the security state related information is retrieved from LMSSF-S and forwarded to provision the Decryption-POI (D-POI) over LI_X1_CR (The D-POI might also need to be provisioned, not shown.).

As discussed in clause 6.3.1.3, if the LMSSF has access to decryption keys (e.g. provided by the SEAF over LI_X1_CR as defined in solution #3.1) this would allow the LMSSF to process protocols where also the handshake is encrypted in a more efficient manner by decrypting the handshake in real-time. This avoids the need to later decrypt this, thus also avoiding race-conditions occurring when LI is activated.

Some mechanism is needed to determine when information captured at the LMSSF can safely be erased.

EXAMPLE: Determining that a captured handshake pertaining to a specific UE/user can be erased would in most cases correspond to the SPDF detecting a new security protocol session between the same UE/user and the same STF, using the same protocol, and/or, if it is detected (e.g. at the SEAFs) that the KSF has provided an updated key for the same (user, STF)-combination.

6.5.1.3 Evaluation

The evaluation follows from the individual evaluations of solution #1.1, #1.3, and #2.1. A first issue highlighted in these evaluations is the need for this LMSSF-S to store information (on the order of hundreds of kB) per security protocol handshake and the trade-off between on one hand storage requirements, and, on the other hand, security (giving the LMSSF access to decryption keys, even before LI is activated).

A second issue is the need for processing capacity and avoiding unnecessary delays, in particular when processing security protocol handshakes as discussed in relation to solution #1.3. Thus, in terms of placement of the LMSSF, it could be suitable for co-location with a UPF (vUPF in the roaming case).

The trade-off between avoiding race conditions when LI is activated, and the potential increased security by not transferring keys to the LMSSF in advance ought to be analysed more deeply.

There are no additional standardization concerns since defining the LMSSF functionality can be done entirely within the responsibility of 3GPP SA3-LI.

6.5.2 Solution #4.2: Decryption POI (D-POI)

6.5.2.1 Introduction

Central to the present document is an LI-function responsible for performing the actual decryption of xCC before it is handed over to the LEMF. The main task of this entity is to be the "host" of solution #1.4, the security processing state machine.

6.5.2.2 Solution details

The functionality is basically of POI-type but extended with decryption functionality so it is natural to conceptually view it as special type of CC-POI, denoted a Decryption-POI (D-POI). It is therefore also natural to consider co-location with a UPF CC-POI. Alternatively, one could split the functionality into two parts: a conventional UPF CC-POI that sends raw encrypted security protocol PDUs to the D-POI. In the latter case, the D-POI could also be co-located with the MDF (i.e. MDF3).

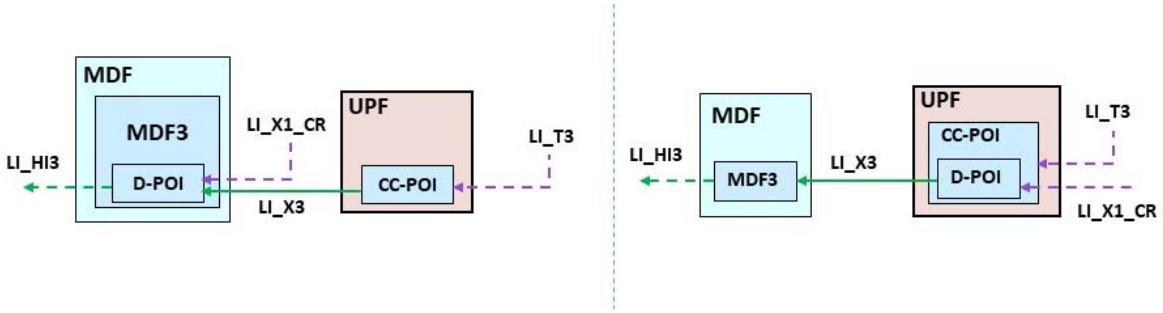

Figure 6.5.2.2-1: Two options for D-POI placement. The D-POI would in both cases need to receive cryptographic parameters necessary for decryption over the LI_X1_CR interface.

NOTE: The D-POI could (and probably would) also receive auxiliary security parameters from the aforementioned SHIFF, which in the rightmost option above would take place over a conceptual "CC-POI internal" interface, which is not shown in figure 6.5.2.2-1.

6.5.2.3 Evaluation

Placing the D-POI with the MDF could have one advantage, namely that the MDF is probably located in an even more secure location than the UPF. However, the CC-POI of the UPF is assumed to be very well protected as-is so the advantage might not be significant. It might also seem as if placing D-POI in the MDF would limit the need for confidentiality on the interface between the UPF CC-POI and the D-POI. This is likely a false assumption since it could be used to detect targets by comparing traffic going in and out of the UPF interfaces. A disadvantage of having D-POI in the MDF might be that current MDF implementations are most likely not dimensioned to run per-target decryption at line speed and distributing the decryption task across UPF POIs is more likely to scale.

Regarding performance, when there are several active intercepts for many sessions, it obviously creates computational load on the D-POI due to the need to decrypt many sessions simultaneously. On the other hand, it seems unlikely that such a large number of simultaneous intercepts would be active that it pushes the performance requirements above what would be expected from a typical webserver handling a (very) large number of, say, HTTPS-connections.

Since the D-POI will obtain access to decryption keys, it needs to provide sufficient assurance, see solution #5.1 below.

Defining the D-POI function can be handled entirely within 3GPP SA3-LI responsibility.

6.6 Solutions for security assurance

6.6.1 Solution #5.1: D-POI SCAS

6.6.1.1 Introduction

The D-POI needs to handle key material and this puts requirements on secure storage and handling. All components of the LI-system already have very stringent requirements on security in order to ensure that the LI-system in no way increase the risks to the overall CSP network. In practice, this implies that the qualitative requirements are already as high as the most security critical components of the network.

6.6.1.2 Solution details

There is already ongoing work in 3GPP SA3-LI to produce Security Assurance Specifications (SCAS) for implementation of LI components. The solution proposes that an explicit SCAS for the D-POI function is included in

the scope of the existing work to ensure that a systematic analysis of the exact requirements for a D-POI is well documented and can be used to support implementations.

6.6.1.3 Evaluation

The solution depends only on specification work within 3GPP SA3-LI scope and is judged to be feasible and effective.

NOTE: As presented above, there was no absolute need that the LMSSF of solution #4.1 also has access to keys. However, if it turns out that LMSSF implementations would be greatly simplified if enabled to decrypt (parts of traffic), then SCAS work for the LMSSF would also become relevant.

6.7 Solutions for UE-to-UE protected communication

6.7.1 Solution #6.1: Interception of UE-to-UE E2E protected communication

6.7.1.1 Introduction

This solution is more of a feasibility analysis intended to get an understanding what would need to be additionally required to use solution for LI of UE-to-STF encrypted communication also for UE-to-UE encrypted communication.

The present document makes no assumption on explicit LI-support in the STF (or the UE). The only general requirement used so far is usage of CSP-provided keys. For the purpose of the discussion below, it is in analogy assumed that the communicating UEs do not provide any LI-support, only that they use CSP-provided keys.

6.7.1.2 Solution details

Here AKMA is probably an unlikely option for direct UE-to-UE usage. The set of feasible approaches would rather rely on a central key management server such as that of the IMS Media plane security framework of TR 33.828 [12], specifically the "ticket based e2e" solution of clause 7.1 of TR 33.828 [12]. Additionally, SRTP (see IETF RFC 3711 [13]) could be a more likely security protocol to be in use here.

Besides the use of CSP-provided keys as above, the solution is additionally based on solution #1.5, "Cipher suite profiling", limiting the cipher suites which do not depend on other keys than those provided by the CSP.

Given the above setting, the solution to KI #1.2 (obtaining keys) could be implement using an LI-interface of the key management server, e.g. similar to TS 33.107 [2], clause 7A.7.

The LMSSF of solution #4.1 would be directly applicable to obtain other IRI from UP traffic, and, if necessary, provide storage functions enabling mid-session intercept. The D-POI function of solution #4.2 can be provisioned by the necessary keys to perform the actual decryption.

6.7.1.3 Evaluation

Under very similar assumptions as those used in the UE-to-STF case, the same solution concepts would seem in principle applicable also to UE-to-UE encrypted sessions and could be handled mainly within 3GPP standards.

The same consideration for the cipher suite profiling of solution #1.5 obviously applies and would again need to be adopted by UE manufacturers.

The solution outlined above only considers the technical feasibility. As discussed in KI #6.1, there could be a need to also consider implications on trust models, a topic that is left FFS.

7 Conclusions

7.1 Impact of solutions

7.1.1 General

In the following, the impact of the solutions is analysed under the assumption that the LI-system already has an equal, or higher security assurance level than any other component of the PLMN. In roaming, this is assumed to hold both in HPLMN and VPLMN, i.e. the LI-system of the VPLMN is at least as secure as the most secure part of the VPLMN, outside the LI-system, and similarly for the HPLMN.

7.1.2 Security

7.1.2.1 Subscriber privacy

In the non-roaming case (with STF located at an external DN) there is minimal impact on the subscriber's privacy. The HPLMN already has access to the CSP-provided keys, so threats of those keys being misused for "mass surveillance" or other unauthorised activities is not affected by the keys also being available in the LI-system. There could be a potential issue related to the need to abstain from usage of certain cipher suites (see KI #1.5). By updating the CSP-keys regularly, the only residual threat would be a compromise of the long-term key in the (U)SIM.

NOTE: At the time of the development of the present document (early 2026), within 3GPP this was not acknowledged to be a significant threat.

In the roaming case with keys being made available also the VPLMN, the threat surface for attacks against the keys arguably increases. At the same time, a proper LI-solution for roaming with encrypted traffic avoids the need to define S8HR/N9HR-like solutions where encryption is simply not enabled. From the subscriber point of view, this is likely an *increase* in security compared to the current situation.

7.1.2.2 Implementation and security assurance

While the present document recommends conducting studies on security assurance for new LI-components such as D-POI and LMSSF, considering that the LI-system already has very stringent requirements on secure implementation, additional requirements, if any, are highly unlikely not to be manageable.

7.1.3 Standardisation

Except for the following two aspects, all the solutions described lie fully within the scope of 3GPP SA3-LI TSs:

- The cipher suite profiling of solution #1.5 would perhaps benefit from being defined in IETF. However, it also seems possible to handle it as adaptations, done within the scope of 3GPP SA3.
- To support roaming, there is need for a new inter-PLMN interfaces and SBA-extensions to allow the VPLMN to get notified about HPLMN key management events and to receive keys. This can be done within 3GPP SA3 and 3GPP CT-groups and, perhaps, 3GPP SA2.

7.2 General conclusions

The following solutions are always necessary to be in place in order to address key issues that are general and do not depend on support for roaming or mid-session intercept (KI #1.1 to #1.5):

- #1.1 SPDF.
- #1.2 Use of AKMA or equivalent.
- #1.3 SHIFF.
- #1.5 Cipher suite profiling.

- #4.2 D-POI (or some other, equivalent implementation of solution 1.4).
- #5.1 D-POI SCAS (implied by solution 4.2).

When adding roaming, the following solution also becomes necessary:

- #3.1 Extended KSF and SEAF functionality.

The additional complexity to add roaming comes mainly from the need to define new inter-PLMN interfaces.

Finally, to add mid-session intercept it is additionally necessary to consider the solution:

- #4.1 LMSSF (which implements solution #2.1).

This is also where most of the complexity is added. It is therefore possible to identify three levels of complexity:

i. Support for non-roaming only and no support for mid-session intercept.

ii. Adding roaming.

iii. Adding mid-session intercept.

Neither solutions #3.2 (Key hierarchy and key separation), nor #6.1 (Interception of UE-to-UE E2E protected traffic) are strictly necessary as basic functionality for any of the choices but rather gives additional security and coverage of more use cases. For examples solution #3.2 could relax the need for inter PLMN-trust in the roaming case. Thus, the complexity and amount of standardisation work needed depend mainly on the choice whether to support mid-session intercept.

In the following sub-clauses, conclusions for individual solutions are summarised. Except where otherwise noted, the solutions can be handled entirely within 3GPP SA3-LI specifications.

7.3 Conclusion on solution #1.1: Security protocol detection

The need for this solution is general and the complexity is small.

7.4 Conclusion on solution #1.2: Use of AKMA (or equivalent)

The need for a solution that enables the CSP to provide keys is generic. AKMA already exists, including an LI-solution.

7.5 Conclusion on solution #1.3: Handshake interception

The need for this solution is general and the complexity is mainly related to the usage of this function in support of mid-session intercept. If only sessions initiated after LI-activation is to be intercepted, the only issue is to provide an implementation with line-speed performance. A stand-alone SHIFF is only necessary to handle mid-session intercept, whereas in other cases the SHIFF could just as well (or even more suitably) be integrated with the D-POI. Integration with D-POI is also more suitable to handle intercept of protocols where the handshake itself might be encrypted.

7.6 Conclusion on solution #1.4: Security processing state machine

The need for this solution is general and the complexity is small. The D-POI concept of solution 4.2 appears to be a feasible approach, with main consideration also here being the line-speed performance.

7.7 Conclusion on solution #1.5: Cipher suite profiling

The need for this solution is general and the complexity is small. The solution needs to be implemented as part of 3GPP crypto profile specifications handled by 3GPP SA3.

7.8 Conclusion on solution #2.1: Security state mirroring

This solution is required for mid-session intercept and has higher complexity than any other solution in the present document. The preferred way to implement it is via an LMSSF-like concept as described in solution #4.1.

7.9 Conclusion on solution #3.1: Extended KSF and SEAF functionality

This solution is made necessary by roaming support. The complexity is low or moderate and can be integrated with SBA, but requires specification work outside 3GPP SA3-LI, including 3GPP SA3, 3GPP CT-groups and possibly 3GPP SA2.

7.10 Conclusion on solution #3.2: Key hierarchy and key separation

This solution is a non-mandatory feature that could relax trust between HPLMN and VPLMN in roaming scenarios. There is a trade-off in terms of using the same key for both encryption and integrity vs using separate keys: in the first case, the LI system gets more robust means to verify that the traffic is generated by the target, but it also opens up for a "buggy" (or malicious) LI-system to generate traffic appearing to come from the target.

The solution is straight-forward but requires specification work in 3GPP SA3.

7.11 Conclusion on solution #4.1: LI Mirror Security State Function (LMSSF)

This solution is the concrete proposal for implementation of the security state mirroring and has high complexity. Though experience with the LMISF defined for S8HR/N9HR suggests that this type of solution is feasible, it needs to be understood that the complexity in implementing an LMSSF is much greater, in particular for security protocols using UDP transport and which encrypts (part of) the handshake and where additional buffering might be needed. More in-depth discussion on the implementation issues can be found in annex A.5.

7.12 Conclusion on solution #4.2: Decryption POI (D-POI)

The need for this solution is general and the complexity is moderate. Also here, the main consideration is the need for line-speed decryption capability.

7.13 Conclusion on solution #5.1: D-POI SCAS

Work on SCAS for D-POI is necessary to avoid that the D-POI does not pose threats to subscribers' encrypted communication. However, no main hurdles to complete such as SCAS is foreseen.

7.14 Conclusion on solution #6.1: Interception of UE-to-UE E2E protected communication

This is an optional solution that might be useful to intercept encrypted user-to-user services, e.g. the IMS DC (Data Channel). As long as the cryptographic keys are provided by the CSP, the existing handling of IMS media plane security (see TR 33.828 [12]) appears feasible to use as a baseline for such services. Some issues related to roaming (if supported) and associated trust models likely need to be studied further.

8 Complete LI-architecture summary

8.1 General

Figure 8.1-1 summarizes the complete architecture with support for both roaming and mid-session intercept.

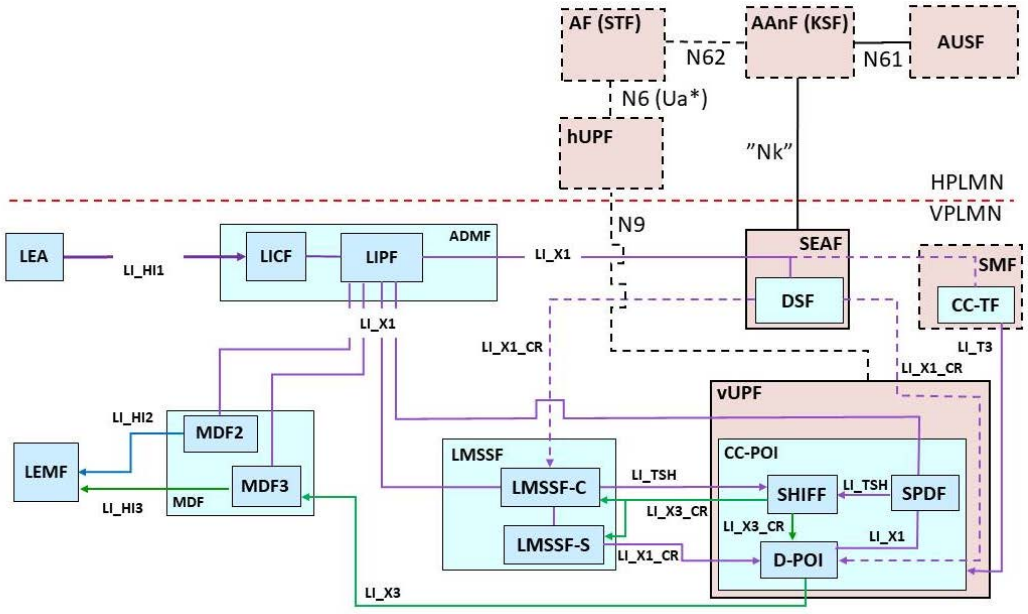

Figure 8.1-1: Architecture summary supporting roaming and mid-session intercept. To handle protocols that encrypt (parts of) the handshake, the SHIFF is more suitably integrated with the D-POI, except in the case of mid-session intercept, see text for discussion.

A summary of a potential solution for provisioning and handling of intercepts could be as described in the following clauses.

8.2 Normal provisioning

For intercepts of new security protocol sessions (i.e. sessions that have not already started before LI is provisioned from the LIPF):

- For each inbound roamer who attaches in the VPLMN, the DSF in the SEAF subscribes to key management notifications from the HPLMN over the "Nk" interface as described in solution #3.1.
- When the DSF of the SEAF receives a new or updated key from the KSF in the HPLMN, it notifies the LIPF of this over LI_X1 (Management).
- As a new intercept is activated, the LIPF can provision the CC-TF of the SMF with the cryptographic keys, enabling the CC-TF to include them in the LI_T3 trigger to the CC-POI of the vUPF, resulting in the keys also being made available to the encryption related functions co-located with the CC-POI, in particular the D-POI. Whether or not decryption would need to take place at the D-POI depends on whether or not the SPDF detects that security is activated for the session.
- To avoid race conditions in completing the triggering of the D-POI, the SPDF (upon detection of a security protocol being taken into use) could trigger the SHIFF to capture the handshake over LI_TSH, while the D-POI is still in the process of being configured to start the intercept. The information (auxiliary security parameters etc.) from the handshake would then be provided from the SHIFF to the D-POI over LI_X3_CR. If a security

protocol is in use that allows (parts of) the handshake to be encrypted, the SHIFF would in reality need to be integrated with the D-POI.

- As a further option, if the DSF of the SEAF receives an updated key from the KSF of the HPLMN after the intercept is first activated, the DSF could continuously supply the D-POI of the vUPF with these new keys over LI_X1_CR as they become known, ensuring that the intercept can proceed even if they keys used are dynamically updated during the session.

8.3 Mid-session intercept

8.3.1 Preparation

- It is also here assumed that the DSF in the SEAF is already subscribing to key management notifications as descried above.

- It is additionally assumed that the LMSSF and SPDF has been provisioned to detect any security protocol handshake of potential interest for interception as described in solution #1.2.

- When a potentially interesting security protocol handshake is detected by the SPDF, it triggers the SHIFF as discussed in solution #1.3. This in turn results in a query-response exchange with the LMSSF-C whether to intercept the handshake as discussed in solution #4.1. (This occurs even if there is no LI_T3 triggering received from the CC-TF SMF.) Thus, in the case of mid-session intercept, the need for having SHIFF-functionality which is independent of the D-POI holds regardless of whether or not the handshake can be encrypted.

- Assuming this is the case, the handshake is captured and stored at the LMSSF-S.

- There is here an option that if the handshake is encrypted and keys are available for the security protocol at the DSF in the SEAF, those keys could be made available to the LMSSF at this point (over LI_X1_CR), enabling the LMSSF to explicitly extract relevant information from the handshake already at his point, rather than to store encrypted raw data.

- If there are further handshake messages occurring later during the session, these are also captured at the SHIFF and forwarded to the LMSSF-S, enabling it to provide and maintain up-to-date cryptographic context information.

8.3.2 Intercept activation

If there is later LI activation for a specific target, it can be determined if there are ongoing security protocol sessions for which relevant information can be retrieved from the LMSSF-S and provisioned to the D-POI (alongside keys from the DSF of the SEAF). As a first step, the D-POI might need to decrypt parts of the previously captured handshake obtained from the LMSSF-S, before being able to start delivering actual decrypted xCC.

Annex A:
Illustration of scenarios

A.1 Introduction

For better understanding of the technical implementation of the solution concepts considered in clause 7, some concrete scenarios for the case that AKMA is used to handle CSP-provided keys, together with various flavours of the TLS/DTLS protocols (IETF RFC 5246 [14], IETF RFC 8446 [15] and IETF RFC 9147 [16]) are provided below. This is motivated by the fact that the AKMA framework is the main solution available in 3GPP specifications and the fact that TLS/DTLS are the main protocols for which AKMA-specific profiles are defined.

A.2 General considerations for AKMA with TLS variants

The AKMA protocol itself is assumed well known, otherwise refer to TS 33.535 [11] for details. For basic understanding of the AKMA-associated LI solutions, refer to TS 33.127 [4]. A short overview of the TLS and DTLS protocols is given in each of the sub-clauses below.

NOTE 1: In the case of DTLS, only version 1.3 defined in IETF RFC 9147 [16] is included as illustration.

The security protocol for which AKMA provides cryptographic keys is in AKMA-terminology referred to as the *Ua* protocol*. When Ua* is based on TLS/DTLS, there are two main options as defined in TS 33.222 [9]:

I. Authenticating the AKMA AF (the STF) is done using a server certificate and establishing a (D)TLS-tunnel. The subscriber/UE then authenticates inside this tunnel using the AKMA provided key K_{AF}.

II. Using the AKMA provided K_{AF} to *mutually* authenticate between UE and AF *and* to establish a (D)TLS-tunnel based on the same K_{AF}.

Option (I) does not allow for decryption of the (D)TLS-tunnel, since the keys used to establish that tunnel are independent on K_{AF}.

NOTE 2: If option (I) is used, it is difficult to even detect (at the UPF) that AKMA is in use. Although the UE indicates AKMA-usage to the AF by including the string "3GPP-AKMA" in the User Agent HTTP header, this header is sent inside the TLS tunnel.

Therefore, assumption on use of option (II) is needed in order to produce a relevant use-case which was assumed in solution #1.5. However, option (II) has sub-options depending on whether the (D)TLS-encryption keys depend only on K_{AF} (i.e. "preshared-key-only"), or, whether they also depend on a Diffie-Hellman exchange. As discussed in KI #1.8, the latter is problematic from decryption point of view, so it needs to be assumed that a preshared-key-only cipher suite is in use.

NOTE 3: The set of currently 3GPP-standardized (D)TLS cryptographic profiles do *not* mandate use or even support for preshared-key-only cipher suites.

NOTE 4: Even if the UE indicates a preshared-key-only cipher suite, specification [9] mandates that the UE also includes cipher suites which are not preshared-key-only. Nothing prevents that the TLS-server (the AF) chooses one of the latter in which case AKMA keys will not be used.

In the sequel, the following implementation model has been adopted. The general process for: (a) LI activation prior to encrypted session start, and, (b) LI activation with established encrypted session, follow the outlines of solution #1.1, #1.3, #1.4, in case (a), and the same solutions in combination with solution #2.1 for case (b). The use of AKMA is common to all scenarios, and the LI handling is according to solution #1.2 and #1.5.

The functionalities of solutions #1.1, #1.3, #1.4 are assumed implemented in the D-POI concept as described in solution #4.2. The additional mirroring functionality needed to support scenario (b) is assumed implemented according to the LMSSF concept defined by solution #4.1.

The illustrations in the clauses below do not go into details of LI-provisioning or triggering and the outlines of solutions #3.1, #4.1, and #4.2 apply to this end.

A.3 Use of AKMA with TLS 1.2

A.3.1 TLS 1.2 overview

TLS 1.2 [14] operates by transforming SDUs, both from the application layer, and from TLS own control layer(s), into TLS PDUs, known as *TLS records*. TLS can be viewed as consisting of four sub-protocols:

- Handshake, exchanging and negotiating security related parameters,
- ChangeCipherSpec, serving to switch the protection from "off" to "on" state,
- Alert, handling errors and notifications, and,
- Application, carrying the protected application layer SDUs.

TLS records have a header containing information to identify the respective sub-protocol and a payload, usually in protected format. The protected payload conveys ECSI information in-band. The full CSI comprises the ECSI and a 64-bit sequence number starting at zero. The sequence number is never exchanged between the endpoints; since TLS runs over reliable transport, there is no need to do so. The figure below shows TLS placement within the stack and the format of TLS 1.2 records. The format of the record fragment depends on the chosen cipher suite, and figure A.3.1-1 shows the case of a typical AEAD cipher suite. The type field in the inner handshake header indicates the type of the handshake message.

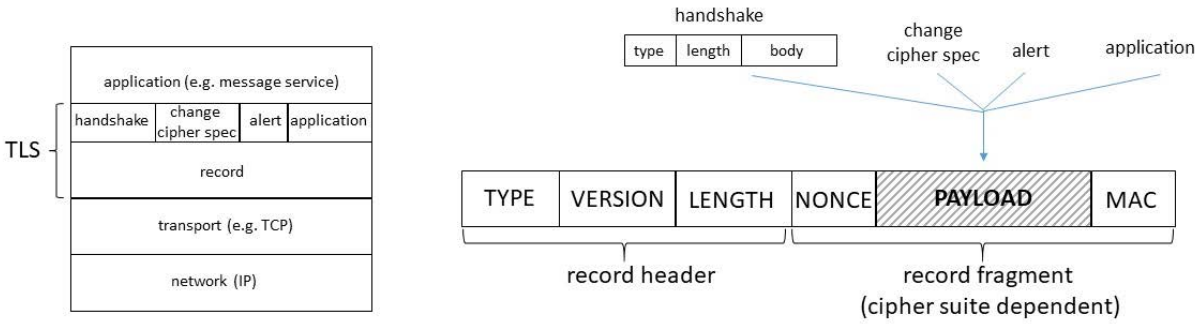

Figure A.3.1-1: TLS within the stack and the TLS 1.2 record format for protected records where the greyed part is encrypted. Only format details for the TLS Handshake protocol is shown. The NONCE serves as ECSI.

TLS 1.2 internally derives keys for record protection from a so-called pre-master key. When used with AKMA, this corresponds to an AKMA K_{AF} key.

A.3.2 LI activation prior to encrypted session start

A.3.2.1 Security protocol detection at SPDF

TLS uses standard TCP ports for various applications, e.g. port 443 for HTTPS. Besides using port number as indication of TLS usage, the record format above (in particular the header) can be used to recognize TLS usage. Since only TLS 1.2 sessions based on AKMA keys are of interest, a filtering is also done by SPDF inspecting TLS 1.2 initiations by UEs indicating AKMA key usage. As here the use of option II (see clause A.2) is assumed, the following means are available:

- The UE sends the hostname of the AF using the server_name extension to the TLS 1.2 ClientHello. While this (as such) does not imply AKMA, the AF hostname is useful if the provided K_{LI} decryption key is the AKMA anchor key (K_{AKMA}) rather than the AF-specific key K_{AF}, since derivation of K_{AF} (depending on the AF host name) is then necessary.

- The PSK-identity field of the ServerKeyExchange handshake message will have the value "3GPP-AKMA" and the corresponding field of the ClientKeyExchange message will have the value "3GPP-AKMA" followed by the A-KID if (and only if) AKMA is taken into use.

Contents of the TLS ClientKeyExchange handshake message can be used to further confirm use of TLS with AKMA, see clause A.3.2.3.

A.3.2.2 Obtaining key management xIRI by use of AKMA

The IRI-POI of the AKMA AAnF (corresponding to the KSF), via the DSF in the SEAF, provides the D-POI with xIRI corresponding to the key K_{LI} and key identifier KID associated with the target, i.e. the AKMA-defined A-KID. The key K_{LI} is either the AKMA anchor key K_{AKMA} or the key for a specific AF, K_{AF}. K_{AF} is in this case derived from K_{AKMA} according to TS 33.535 [11] as

K_{AF} = KDF(K_{AKMA} , FQDN of AF || Ua* protocol identifier || ….)

where KDF is the key derivation function defined in TS 33.220 [8]. Hence, knowledge of K_{AKMA} allows the D-POI to derive keys for any AF and using any 3GPP-defined Ua* protocol. Under the assumptions made (use of TLS 1.2 with option II as described in clause A.2), the Ua* protocol identifier will be a five-octet value of form (0x01,0x00,0x01,yy,zz), where the last two octets encode the proposed TLS 1.2 cipher suite to be used.

A.3.2.3 Handshake interception at SHIFF

For the concrete example of TLS 1.2, the full set of xIRI can be found in the ASN.1 payloads attachment to TS 33.128 [5] and includes:

- Selected TLS PRF algorithm and cipher suite,
- TLS compression algorithm,
- Client (UE) and server (AF) random values,
- TLS session identifier, and,
- The client's proposed and server's accepted TLS extensions.

The above values are thus obtained by interception of the TLS handshake by the SHIFF. The PSK-identity field of the ClientKeyExchange handshake message can be used as a verification of the use of AKMA key material, since in this case, this field will have the format "3GPP-AKMA" followed by the A-KID.

A.3.2.4 D-POI security processing state machine

A.3.2.4.1 Deriving initial cryptographic context

TLS 1.2 uses 64-bit sequence numbers (one for each direction; client-to-server and server-to-client) as basis for cryptographic synchronization information. These are always initialized to zero when the session starts.

The other information that needs to be derived to initialize the cryptographic context is the cryptographic keys. When AKMA is used with TLS 1.2, the TLS pre-master secret is initialized from the AKMA K_{AF} key, obtained according to clause A.3.2.2. The TLS master secret is derived from the pre-master secret and the client and server random values in the cryptographic context as defined in IETF RFC 5246 [14], clause 8.1. Finally, the encryption and data integrity keys and the cipher IV are derived from the master secret according to IETF RFC 5246 [14] clause 6.3.

The D-POI is now set up to receive and process protected (encrypted) messages sent between UE and STF.

A.3.2.4.2 Processing (decrypting) of xCC

This follows the general outline of clause 4.3.3, keeping in mind that TLS 1.2 generally uses the MAC-then-encrypt processing order, or the order defined by the AEAD algorithm in use. The needed ECSI (e.g. nonce) is extracted directly from the TLS record.

A.3.2.4.3 Updating cryptographic context

The TLS sequence number (associated with the direction of the most recently processed TLS record) increments by one for each subsequent TLS record.

A.3.3 LI activation with established encrypted session

A.3.3.1 Security protocol detection at SPDF

In preparation for a future LI-activation for which a TLS 1.2 connection has already been set up, the SPDF detects TLS 1.2 usage as described in clause A.3.2.1.

A.3.3.2 Obtaining key management IRI by use of AKMA

The principles of clause A.3.2.2 applies, observing that when an intercept is activated, the AF host name obtained from the handshake and stored at the LMSSF is necessary to derive the AF-specific key K_{AF} if the provided K_{LI} decryption key obtained from the SEAF DSF is the AKMA anchor key.

A.3.3.3 Handshake interception at SHIFF and LMSSF

TLS 1.2 handshakes are not encrypted, again making this task relatively straight forward. All xIRI from the TLS 1.2 handshake are now assumed provided by the SHIFF to the LMSSF for storage, collected as described above. Hypothetically, LI-activation might occur while (part of) the handshake is not yet complete so that it can be directly captured by the CC-POI of the UPF, but it appears simpler to always rely on the LMSSF.

There remains one active task for the LMSSF after the handshake, maintaining two (per-direction) counters for the TLS 1.2 session, basically "packet counting" functionality available at the UPF POI. This task can however in most cases be seen as an optional feature, see discussion in clause A.3.3.4.1.

A.3.3.4 D-POI security processing state machine

A.3.3.4.1 Deriving current cryptographic context

This is handled by first deriving the initial cryptographic context as discussed in clause A.3.2.4.1. The remaining task in order to establish the current context is to decide the TLS-internal 64-bit sequence numbers. This is greatly facilitated if the UPF D-POI and LMSSF has monitored the TLS connection and kept up-to-date values for the sequence numbers as discussed above. If a non-AEAD cipher suite is used, the TLS sequence numbers are only needed to verify integrity of TLS records. But since verifying integrity does add robustness to the interception and since AEAD cipher suites are quite likely to be used, determination of the sequence numbers ought to be pursued.

A.3.3.4.2 Processing (decrypting) xCC

This step follows the description of clause A.3.2.4.2.

A.3.3.4.3 Updating cryptographic context

This step follows the description of clause A.3.2.4.3.

A.4 Use of AKMA with TLS 1.3

A.4.1 TLS 1.3 overview

The notable differences between TLS 1.3 [15] and TLS 1.2 are mainly the following:

- Only AEAD cryptographic algorithm are defined for use.

- Parts of the Handshake protocol is also encrypted.
- The client (UE) can start to send protected application data interleaved with the Handshake. This is known as *early data*.
- There is no ChangeCipherSpec sub-protocol, the point at which security is switched "on" is always well-defined.
- There is both an outer record header, as well as an inner header:
  - All encrypted records use TYPE = 23 in the outer header, indicating the TLS Application protocol.
  - The inner header contains the "true" type and is found inside the encrypted record fragment.
- A mechanism is included which allows the keys for the Application protocol to be dynamically "refreshed" during an ongoing session.

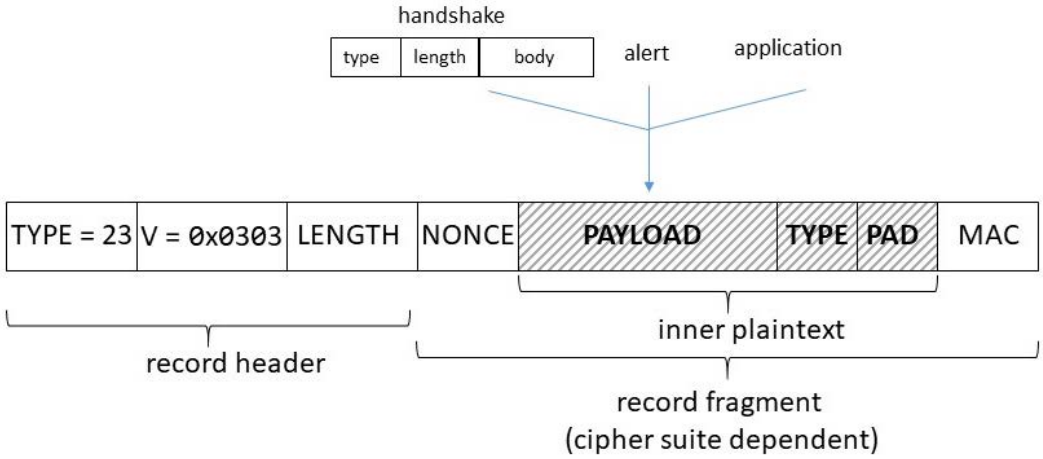

Figure A.4.1-1: TLS 1.3 record format for protected records. The greyed part is encrypted where PAD includes padding (if any). The format details for Alert and Application protocols are not shown.

Like TLS 1.2, TLS 1.3 also internally derives keys for record protection. However, there is a much larger number of keys derived and used for different purposes. When used with AKMA, the base key for all the internal keys corresponds to an AKMA K_{AF} key.

NOTE: There is currently ongoing work in IETF around TLS 1.3 which could lead to additional considerations. This includes the so-called "encrypted Client Hello" and the migration to post-quantum secure algorithms. These work items are outside the scope of the analysis below.

A.4.2 LI activation prior to encrypted session start

A.4.2.1 Security protocol detection at SPDF

The discussion for TLS 1.2 in clause A.3.2.1 applies. However, the initial messages (ClientHello and ServerHello) use TLS version 1.2 in the TLS message header. To determine that TLS 1.3 is being negotiated, it is necessary to look inside the TLS Extension field of these messages. Further, to determine that TLS 1.3 is being used with AKMA, it is necessary to inspect the PresharedKey extension to the ClientHello, see clause A.4.2.3.

A.4.2.2 Obtaining key management xIRI by use of AKMA

The description of clause A.3.2.2 applies (mutatis mutandi) also for TLS 1.3.

A.4.2.3 Handshake interception at SHIFF

In the case of TLS 1.3, the SPDF and SHIFF can detect use of TLS 1.3 but cannot in general determine which (encrypted) messages that are part of the Handshake (carrying IRI) or part of early data or application. This is because

all encrypted records have the outer record header indicating Application data, even if it contains Handshake messages. This determination and extraction of xIRI can only be done after decryption at the D-POI, so the first step is to derive the decryption keys needed for decryption of the Handshake (the *_handshake_traffic_secret of table A.4.2.4.1-1, see below in clause A.4.2.4.1). In other words, this means that the SHIFF functionality needs to be tightly integrated with the D-POI.

The encrypted part of the TLS Handshake protocol is well-defined. The first encrypted message is the EncryptedExtensions handshake message (sent from the AF to the UE) and all remaining TLS Handshake messages (up to the final Finished message from the UE) are encrypted.

The following xIRI from the TLS 1.3 handshake are needed (or are generally of relevance for LI). The full set of xIRI can be found in the ASN.1 payloads attachment to TS 33.128 [5] and includes:

- The set of pre-shared keys (key identities) offered by the UE (in the AKMA case, there ought normally to be just one),
- The pre-shared key identity selected by the AF,
- Cipher suite and key-derivation function (offered and selected),
- Random values (nonces) from the UE and the AF,
- The UE-proposed and AF-selected TLS extensions, and,
- Certificates associated with UE and AF (if available; use of AKMA does not require certificates).

A difference to the TLS 1.2 case is that Ua* definition for TLS 1.3 states that the UE now indicates use of AKMA, not in a ClientKeyExchange handshake message, but rather as part of a PresharedKey extension to the ClientHello. The format is however the same, the PSK identity field is set to "3GPP-AKMA" followed by the A-KID.

TLS 1.3 allows so called early data to start to flow, interleaved with handshake messages, whose handling is discussed in clause A.4.2.4.

A.4.2.4 D-POI Security processing state machine

A.4.2.4.1 Deriving initial cryptographic context

Initial context here refers to the values which are valid before any early data or encrypted handshake messages (see clause A.4.2.4.2) have been exchanged between UE and AF. That is, values valid at the point where the AF has just sent the TLS ServerHello message.

First, the xIRI listed above can be directly populated into the cryptographic context, along with the TLS sequence numbers (initiated to zero).

The other main piece of information that needs to be derived to initialize the cryptographic context is the cryptographic keys. But in contrast to TLS 1.2, some more computations are needed to derive all the keys needed within the TLS session. First, the external TLS PSK key is set to the AKMA K_{AF} key. The details of the remaining key derivations can be found in clause 7 of IETF RFC 8446 [15], a short high-level summary is provided in table A.4.2.4.1-1 below.

Table A.4.2.4.1-1: The set of derived keys for TLS 1.3. Keys prefixed by * in most cases refer to a pair of keys (one for UE/client and one for AF/server), see text.

Derived key	Derived from
early_secret	external TLS PSK
binder_key	early_secret
client_early_traffic_secret	early_secret
early_exporter_master_secret	early_secret
handshake_secret	early_secret
*_handshake_traffic_secret	handshake_secret
master_secret	handshake_secret
*_application_traffic_secret_N	master_secret
exporter_master_secret	master_secret
resumption_master_secret	master_secret
*_write_key	see text

The derivations also depend on hashed transcripts of the communication between the UE and the AF. Deriving the *_handshake_traffic_secret is a first priority, since those keys are needed to decrypt the Handshake itself, i.e. they are needed to complete the tasks of the SHIFF as discussed above.

The *_write_key in the last row actually corresponds to a client specific key and two pairs of client/server-specific keys:

- The client_early_data_write_key is used for early data and is derived from client_early_traffic_secret.
- The *_handshake_write_key is used for encrypted handshake and is derived from the corresponding *_handshake_traffic_secret.
- The *_application_write_key is used for application data and is derived from *_application_traffic_secret_N.

For each of these five keys, there is also a corresponding "nonce" to be used as part of CSI: client_write_iv and server_write_iv, derived from the same key as the corresponding *_write_key.

The keys *_application_traffic_secret_N can be updated dynamically during a session, see clause A.4.2.4.3.

With all this information in the cryptographic context, the D-POI is now set up to receive and process protected (encrypted) messages sent between UE and STF.

A.4.2.4.2 Processing (decrypting) of xCC

A major difference to TLS 1.2 has already been mentioned: application traffic (early data) can start to flow before the handshake is complete. The keys used to encrypt/decrypt the application traffic are all different but derived from the same basic session key as discussed above. Therefore, the D-POI has to identify which key to use, and this can be done as follows.

If the UE has early data, it includes an (unencrypted) EarlyDataIndication in the ClientHello handshake message. This includes an indication of the maximum size of early data that the UE will send. After this, any TLS Application PDU that follows is encrypted with the corresponding keys. The cipher suite used for the early data is provisioned along with the pre-shared key (in the present case an AKMA key). When the UE has no more early data to send, it indicates this by a special EndOfEarlyData handshake message. Thus, all TLS Application PDU that follows that message use the "normal" session encryption keys and the cipher suite negotiated during the handshake (which in principle *can* be different from the early data cipher suite).

TLS 1.3 only uses AEAD cipher suites. This means that the CSI needed to decrypt (and verify) a TLS 1.3 PDU consists of:

- An ECSI in the form of an in-band "nonce" from the TLS record payload,
- A per-session nonce-value maintained in the cryptographic context (the *_write_iv), and,
- The 64-bit TLS sequence number (also part of the cryptographic context).

A.4.2.4.3 Updating cryptographic context

As in TLS 1.2, the sequence number is incremented by one on each processed TLS record. In TLS 1.3, the session keys can moreover be updated dynamically: a new "generation" key is derived from the current one when a KeyUpdate message is issued (which can occur at any time after the initial handshake is complete). The new key keys are derived as

application_traffic_secret_N+1 = KDF(application_traffic_secret_N, …),

Where KDF is the selected key derivation function, see clause 7.2 of IETF RFC 8446 [15] for details. Therefore, if a KeyUpdate message was just processed, the session keys of the cryptographic context are updated accordingly. If a KeyUpdate occurs, the sequence numbers are also re-initialized to zero.

While it is assumed that the ticket/resumption feature of TLS 1.3 is not used, there is of course no reason why tickets issued by the AF (via TLS 1.3 NewSessionTicket messages) and decrypted at the D-POI cannot be extracted and stored as part of the cryptographic context for potential future use.

A.4.3 LI activation with established encrypted session

A.4.3.1 Security protocol detection at SPDF

Under the assumption that option II discussed in clause A.2 is used, the SPDF can detect AKMA-related TLS 1.3 handshakes as described in clause A.4.2.1, i.e. based port numbers and on the presence of a PresharedKey extension to the ClientHello with the PSK identity field is set to "3GPP-AKMA" followed by the A-KID. This is possible since the ClientHello is never encrypted.

A.4.3.2 Obtaining key management xIRI by use of AKMA

This is handled identically to clause A.4.2.2.

A.4.3.3 Handshake interception at SHIFF and LMSSF

All messages after the ServerHello are encrypted based on (AKMA) keys that the SHIFF (or D-POI) might not yet know. Further, all unencrypted headers indicate the Application type, even if the content is a Handshake message. Additionally, the UE could use the early data option to send encrypted application messages interleaved with the handshake messages. Therefore, the SHIFF needs to capture a more or less "raw" transcript of the rest of the handshake in encrypted format and forward to the LMSSF-S. There still remains an issue to potentially filter out messages that do not relate to the handshake. That task can only be robustly done after decryption (with knowledge of keys). It might of course happen that keys are known at the SEAF, so if made available to the SHIFF, they could be used to simplify the filtering task. However, it could then also result in the collection of application data that was sent prior to LI activation.

NOTE: Whether capturing this early data traffic for later decryption is permitted or not is likely to depend on local regulations. In some jurisdictions, it might be necessary to discard the data after determination that it really was early data, rather than handshake information.

Some xIRI, e.g. cipher suite, the random values supplied by UE and AF, etc., can always be explicitly captured since these are never encrypted during the handshake.

A.4.3.4 D-POI security processing state machine

A.4.3.4.1 Deriving current cryptographic context

The remaining IRI from the TLS 1.3 handshake (that could not be explicitly captured due to encrypted handshake) are provided by the LMSSF as encrypted raw data. Therefore, the D-POI needs to first derive those parts of the cryptographic context that is needed to decrypt the handshake messages provided by the LMSSF, and this corresponds to deriving the initial cryptographic context that was valid when the first encrypted handshake messages were generated.

With reference to table A.4.2.4.1-1, the D-POI first needs to derive the keys: early_secret, handshake_secret, and *_handshake_secret. (The keys binder_key and early_exporter_master_secret could also be derived, though they are not needed for decrypting the handshake). This enables the D-POI to decrypt the remaining Handshake messages (following clause A.4.3.4.2) and to complete the initial cryptographic context (remaining keys, etc.).

It remains to update the initial cryptographic context to reflect the current cryptographic context, in analogy to the description of clause A.3.3.4.1. However, in the case of TLS 1.3 this turns out to be more complex. The reason is that during part of the UE-AF TLS session that has already taken place from the point where the LMSSF captured the TLS Handshake, until LI activation, there can in principle have occurred any number of KeyUpdate Handshake messages, updating the traffic encryption keys. As these are encrypted and reveal no information about being KeyUpdate messages in headers or the like, there is no way for the SHIFF to filter out those: it would need to have captured the entire session. The only other way to handle this is for the D-POI to first try to process a TLS message using the initial *_application_traffic_secret_0 key. If that fails (e.g. as determined by integrity verification failure), the D-POI iteratively derives subsequent keys, *_application_traffic_secret_N, N = 1, 2, … until processing succeeds. This could however be a time-consuming task and to avoid this problem, UE/AF implementations following a policy to avoid using KeyUpdates would be advantageous.

A.4.3.4.2 Processing (decrypting) of xCC

This follows clause A.4.2.4.2 and now consists of two tasks that might require different handling:

1. Early data: if the UE sent encrypted early data that was interleaved with the earlier UE-AF TLS-handshake and that was captured and provided by the LMSSF, these *could* now be decrypted at the D-POI (relative to the initial cryptographic context) *if* permitted by local regulation.

2. Real-time data: decrypting real-time traffic exchanged between UE and AF.

Step 1 would pose challenges in avoiding under-collection if LI-activation occurs concurrently with the real-time exchange of handshake/early data: since it appears that the D-POI has a non-trivial task to complete the processing described in clauses A.4.3.3 and A.4.3.4.1, there could be a need to perform real-time buffering of encrypted UP data that is exchanged between UE and AF until the D-POI "is up to speed" to start the processing of steps 1 and 2.

A.4.3.4.3 Updating cryptographic context

See clause A.4.2.4.3.

A.5 Use of AKMA with DTLS 1.3

A.5.1 DTLS 1.3 overview

DTLS 1.3 [16] builds on TLS 1.3. However, since DTLS runs over unreliable transport, the DTLS Handshake uses retransmissions and acknowledgements. For the same reason, there is more ECSI information included in records to be able to handle reordering and datagram loss. A received DTLS record's ordering within the data stream is determined by:

record_number = epoch_number x 2^{64} + sequence_number,

I.e. a total of 128 bits. The epoch number starts at zero. The epoch number serves an additional purpose since it can be used to identify which of the DTLS internal keys to use to process a received record (more on this below). This is particularly useful considering that the keys can, as in TLS 1.3, be dynamically updated during a session.

NOTE: Using the epoch number to determine the correct key does not increase the probability of failed decryption even if the epoch number happens to be incorrect. Even if the correct key was known via some other mechanism, decryption would still result in error because the correct epoch number is in any case needed as an additional input to the decryption algorithm.

The sequence number also starts at zero and is reset to zero each time the epoch number is increased. The correct value of the record number is needed to process (decrypt and verify) a DTLS PDU since it is used as input to the AEAD cryptographic algorithm. To handle record reordering and loss, a straightforward solution is to include the full record number as ECSI in each DTLS record. However, to save bandwidth, only parts of these quantities are included: the two

least significant bits of the epoch number and the eight (or sixteen) least significant bits of the sequence number. The protected DTLS record format is shown in figure A.5.1-1.

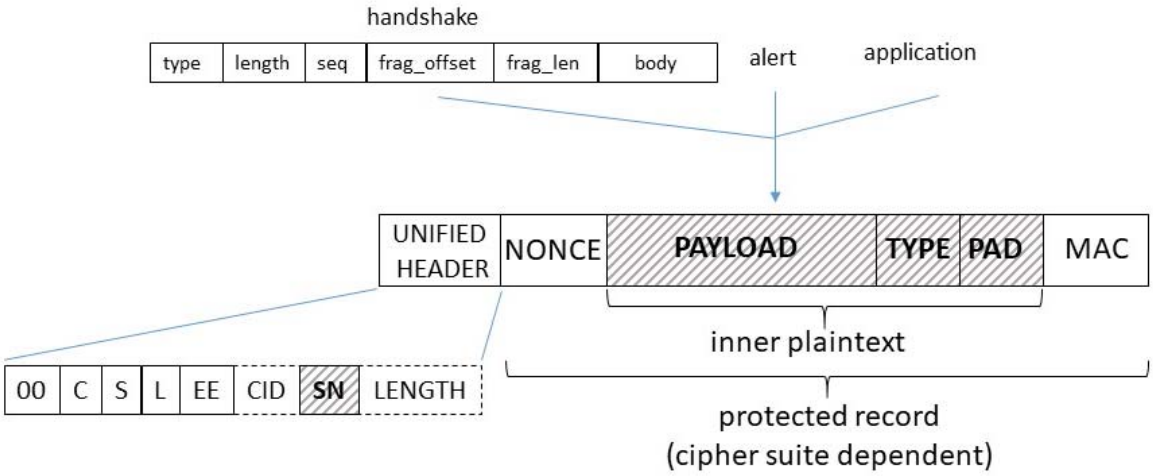

Figure A.5.1-1: DTLS 1.3 record format for protected records. The greyed part is encrypted where PAD includes padding (if any). The format details for alert and application protocols are not shown. See further discussion in the text.

The unified header for a record starts with two zero bits and then follows:

- C, one bit, set if the CID field is present.
- S, one bit, set to 0 if the SN field is one byte, and 1 if it is two bytes.
- L, one bit, set if the LENGTH field is present. (Absence of a LENGTH field means that the record consumes the entire remaining bytes of the lower-level transport datagram.).
- EE, the two least significant bits of the epoch number.
- CID, optional connection identifier, see discussion in text.
- SN, the one (or two) least significant bytes of the 64-bit sequence number.

The EE and SN fields, together with the NONCE in the protected record, constitute ECSI and is thus part of the CSI for the record. Observe that SN is encrypted. This is done to prevent traffic analysis (by linking records having adjacent sequence numbers). However, for obvious reasons, this encryption is independent of the sequence number as it would otherwise create a "chicken-and-egg" problem at the receiving side. This SN-encryption further uses a specific key, see clause A.5.2.4.1.

As in TLS 1.3, it is not possible to determine the actual TYPE of the message (e.g. whether it is a Handshake or Application message) without first decrypting.

DTLS Handshake messages can be quite large and exceed the standard 1500-byte limit on UDP datagrams. Therefore, DTLS has a built-in fragmentation mechanism which works as follows, with reference to the Handshake message format shown in figure A.5.1-1. The sender divides large DTLS Handshake messages by partitioning them into fragments. These fragments are transmitted as separate Handshake messages, all with the same sequence number SN in the "inner", handshake-specific header of the message (the seq field). Further, the fragment's offset (the total number of bytes contained in previous fragments) is included as frag_offset and the length of the fragment is signalled in the frag_len field. This inner header and the rest of the payload are encrypted by the DTLS record layer. This means that in order to complete reassembly of a fragmented Handshake message it is necessary to first decrypt the individual fragments to determine their order.

DTLS has an optional-to-use connection identifier (CID). The intended use of the CID is to simplify and add robustness to the process of identifying which cryptographic context to use to process a received DTLS PDU. When used, the CID is carried in-band in the DTLS headers and allows cryptographic context identification independent of transport parameters such as IP addresses.

DTLS 1.3 defines a special ACK message that is used by the endpoints to acknowledge receipt of record numbers as part of the Handshake.

Since DTLS uses a specific epoch number for early data, the EndOfEarlyData message from TLS 1.3 is not used in DTLS 1.3.

The final aspect of relevance for DTLS is the replay protection. Since reordering can occur due to the transport layer properties, it becomes important to have a mechanism that can protect against replay attacks. This is discussed in more detail in clause A.5.2.4.3.

A.5.2 LI activation prior to encrypted session start

A.5.2.1 Security protocol detection at SPDF

While DTLS defines specific ports for specific applications, UDP port 443 is a common default. Besides this, DTLS detection can inspect the UDP payload, looking for the DTLS header.

A.5.2.2 Obtaining key management IRI by use of AKMA

The description of clause A.3.3.2 applies also for DTLS 1.3.

A.5.2.3 Handshake interception at SHIFF

The handling of DTLS 1.3 in principle follows that of TLS 1.3 described in clause A.4.2.3. As with TLS 1.3, it becomes necessary to integrate SHIFF and D-POI functionality to allow the SHIFF to decrypt and extract IRI. A major additional issue to handle is that the SHIFF needs to be aware of, and identify, DTLS retransmissions and re-ordering during the handshake. Since decryption is needed before reassembly, the SHIFF/D-POI needs to derive the keys used to decrypt the Handshake, i.e. the *_handshake_traffic_secret keys, and buffer fragments until it is able to start processing the complete reassembled message.

Handling of out-of-order messages will be discussed in more detail in clause A.5.2.4.2.

The IRIs to extract from the DTLS 1.3 handshake includes one additional IRI compared to TLS 1.3. This is the connection identity (CID). The full set of xIRI can be found in the ASN.1 payloads attachment to TS 33.128 [5].

A.5.2.4 D-POI security processing state machine

A.5.2.4.1 Deriving initial cryptographic context

The main details follow the discussion for TLS 1.3 in clause A.4.2.4.1. DTLS additionally needs to manage the following information elements.

- The epoch number (part of the CSI for DTLS), which starts at zero.
- The connection identity (if used).
- Two additional keys, the sender_sn_key and the receiver_sn_key. These are used to encrypt/scramble the part of the sequence numbers which are carried in-band as ECSI, see clause A.5.2.4.

As can be seen from the TLS 1.3 keys in table A.4.2.4.1-1, some of the keys are derived from other keys and hashed transcripts of Handshake messages. In the case of DTLS 1.3, any possible fragmentation of Handshake messages needs to be handled, removing fragmentation related header fields and assembling complete messages, before computing the hashed transcripts.

A.5.2.4.2 Processing (decrypting) of xCC

Determining the CSI consists of estimating the 128-bit record number at the D-POI. As discussed above, the epoch number and sequence number start at zero and the latter is reset to zero each time the epoch number is increased.

Recalling that DTLS 1.3 is based on TLS 1.3, the epoch number is then defined and updated during a DTLS 1.3 session, depending on the keys used to protect the PDU. A summary is given in table A.5.2.4.2-1.

Table A.5.2.4.2-1 DTLS Epoch numbers (for epoch values 3 and 4, refer to table A.4.2.4.1-1 for details on the respective key).

Epoch number	Usage
0	All unencrypted messages.
1	Early data.
2	Handshake messages occurring up to, and including, the Finished messages.
3	Messages protected using keys derived from the initial *_application_traffic_secret_N (N = 0). This includes handshake messages after the Finished message.
4..2^{64}-1	Messages protected using keys derived from the *_application_traffic_secret_N (N > 0). That is, the epoch number equals the value N+3.

This implies that even if message reordering can occur, determining the epoch number is usually straightforward except in conjunction to switching from *_application_traffic_secret_N to *_application_traffic_secret_N+1. While this is signalled by a KeyUpdate message (just as in TLS 1.3, see clause A.4.2.4.3), there could be delayed messages from the previous epoch that still have not arrived at the D-POI.

NOTE 1: The DTLS 1.3 specification [16] makes a general recommendation to discard messages from epochs earlier than the "current" one, but from LI point of view, it could make sense to process (decrypt) also those.

To address this, the DTLS unified header includes the least significant bits of the epoch number as ECSI, i.e. as a "hint" to the receiver about the correct epoch.

NOTE 2: As seen in table A.5.2.4.2-1the two bits in the ECSI always uniquely determine the epoch during the Handshake.

In general, if the epoch-bits from the unified header agree with the least significant bits epoch of the previously processed PDU, that epoch number is the correct value also for the current PDU.

NOTE 3: For the two bits to not correctly identify the epoch number, there would need to be at least four key updates which all have been lost or delayed. However, the specification [16] mandates that a new KeyUpdate is not allowed as long as a previous one remains unacknowledged.

Thus, these two ECSI-bits can be used to deduce whether to increase or decrease the epoch number relative to the most recently used value when (successfully) processing a received PDU.

EXAMPLE 1: If the two least significant bits of the just received epoch number is "00", and the corresponding bits of the last used epoch number was "11", i.e. the last used epoch number had format 4e+3, then the current PDU can be correctly processed with epoch number 4e + 4 (and the corresponding key).

EXAMPLE 2: Similarly, if the two least significant bits of the just received epoch number is "11", and the corresponding bits of the last used epoch number was "00", i.e. the last used epoch number had format 4e', then the current PDU is to be processed with epoch number 4e' - 1 (and the corresponding key).

It remains to determine the part of the CSI corresponding to the sequence number. Only the 8 (or 16, depending on configuration) least significant bits of the sequence number are included as ECSI in the DTLS PDU unified header, and in encrypted format. A special decryption algorithm is used for this purpose, whose input depends only on the ciphertext of the DTLS record payload. Therefore, the D-POI can obtain the 8/16 least significant bits of the sequence number as a first step. The D-POI can then extend these bits to the full 64-bit sequence number by analysing the least significant bits of the most recently (and successfully) processed PDU in analogy to the discussion above.

Once the CSI is determined, the remainder of the processing generally follows that of TLS 1.3 as described in clause A.4.2.4.2.

A.5.2.4.3 Updating cryptographic context

Using unreliable transport, the main difference in DTLS is that updates to the cryptographic context needs to be reversible, in case a delayed datagram is received which needs to be processed by an "earlier" version of the cryptographic context.

EXAMPLE: In TLS 1.3, following a KeyUpdate message, the D-POI can safely discard the keys *_application_traffic_secret_N, overwriting them by *_application_traffic_secret_N+1 if one follows then recommendation to discard messages protected by *_application_traffic_secret_N after *_application_traffic_secret_N+1 has been taken into use. However, it might from LI point of view still make sense to be able to process also delayed datagrams protected in dependence on *_application_traffic_secret_N and in that case this key cannot be overwritten.

With respect to CSI, maintaining a "next expected record number" needs to be replaced by "most recently and successfully used record number" (comprising an epoch number and a sequence number) which does not necessarily increase monotonically. If integrity protection is used (which it by default is), updating the CSI of the cryptographic context ought only to be done if the integrity of the PDU was successfully verified.

Thus, in general, the update of the cryptographic context comprises pruning and/or extending a list of cryptographic keys and record numbers which could potentially still be valid for not-yet-received PDUs.

One DTLS-specific consideration is replay-protection. This is mainly a feature intended to protect the actual receiver from replayed information, but the feature also adds robustness to the intercept product. The following describes replay protection at the endpoints (the UE and STF) which could be implemented also at the D-POI. A replay list (or "sliding window") is maintained as follows. The list contains L elements (L is a configurable parameter, chosen depending on how much reordering/loss that it deemed reasonable). The "head" of the list indicates the highest value of the DTLS record number which has been received *and* successfully verified. The following list elements contain the L-1 preceding received/verified record numbers.

EXAMPLE 1: The list is often implemented as a bit-vector of length L. Individual bits in the vector are initially "0" and are set to "1" as PDUs are received and verified. Assuming the highest received/verified record number is R, setting the n:th bit of the vector to "1" indicates that the PDU corresponding to record number R-n has been received and verified. If the next received/verified PDU has record number R' > R, the bit-vector is shifted R' – R steps to the right (with zeros entering from the left) and the first (leftmost, or 0:th) bit is then set to "1".

Upon reception of a PDU with estimated record number R, the following checks are made (in this order):

1. Check if R corresponds to a record number which is too much delayed to fit in the list. If so, the PDU is discarded.
2. If R would fit in the list, but is already recorded as received, the PDU is classified as a replay and is also discarded.
3. Otherwise, the PDU is processed. If integrity verification is successful, R is added to the list, thereby also possibly pruning the list from record numbers that become outdated by now falling outside the list.

From LI point of view, some policy could define if and how to record PDUs that deemed replayed or too much out of order.

EXAMPLE 2: The policy could state to maintain records that are heavily delayed and to discard but log events related to PDU that are deemed as being replays.

A.5.3 LI activation with established encrypted session

A.5.3.1 Security protocol detection at SPDF

The description for DTLS 1.3 in clause A.4.3.1 generally applies.

A.5.3.2 Obtaining key management IRI by use of AKMA

See clause A.5.2.2.

A.5.3.3 Handshake interception at SHIFF and LMSSF

Similar to the TLS 1.3 illustration of clause A.4.3.3, the SHIFF records raw handshake messages and forwards them to LMSSF. The difference to TLS 1.3 is mainly that the handshake messages might be received in the wrong order and reordering and reassembly needs to be postponed until decryption keys are available, e.g. if/when intercept is activated.

A.5.3.4 D-POI security processing state machine

A.5.3.4.1 Deriving current cryptographic context

The first task is to retrieve previously captured Handshake messages from LMSSF, decrypt and re-assemble/re-order them as necessary. The rest of the task is similar to the TLS 1.3 case (clause A.4.3.4.1). In fact, the situation is in some regards simpler in this case:

- The potential issues with KeyUpdate messages that might have occurred between IRI capture at the LMSSF and LI activation are partly alleviated since the epoch numbers imply the number of such key updates that can have occurred, so the two EE-bits of the unified header indicate the correct number of key updates, modulo 4.
- For similar reasons, estimating the full CSI (DTLS record numbers) is to some extent facilitated by the EE and SN fields of the unified header which provide partial information.

A.5.3.4.2 Processing (decrypting) of xCC

This is handled in analogy to the TLS 1.3 description of clause A.4.3.4.2. It is potentially possible that (heavily) delayed DTLS PDUs that were originally sent before LI was activated could still be received at the D-POI after it has obtained capability to decrypt them. Local regulation is assumed to decide how this situation should be handled, i.e. whether or not such PDU falls within scope of the warrant.

A.5.3.4.3 Updating cryptographic context

The discussion in clause A.5.2.4.3 applies also here.

Annex B:
Change history

Change history							
Date	**Meeting**	**TDoc**	**CR**	**Rev**	**Cat**	**Subject/Comment**	**New version**
2025-07	SA3#98-LI	S3i250360				TR skeleton and initial draft	0.0.1
2026-01	SA3#100-LI	S3i260064				Draft agreed at SA3#100-LI	0.0.2
2026-05	SA3#101-LI	S3i260220				Agreed for Change Control	2.0.0
2026-05	SA#112	SP-260375				Submitted to SA#112 Plenary with minor editorial update	2.0.1
2026-06	SA#112					Approved at SA#112 for Change Control	19.0.0

History

Version	Date	Status
V19.0.0	July 2026	Publication