ETSI TR 103 953 V1.1.1 (2024-07)

Cyber Security (CYBER);
Guidelines for TLMSP Usage

Reference
DTR/CYBER-00106

Keywords
cyber security

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

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables 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 Web server (https://ipr.etsi.org/).

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™** and **LTE™** 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.

Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee Cyber Security (CYBER).

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.

Executive summary

Requirements exist for network operators, service providers, users, enterprises, and small businesses, to be able to grant varied (fine grained) permissions and to enable visibility of middleboxes, where the middleboxes in turn gain observability of the content and/or metadata of encrypted sessions. Various cyber defence techniques motivate these requirements. The solutions used in the past have often had one or more shortcomings such as:

- a tendency to break end-to-end aspects of all security mechanisms (not only confidentiality);
- to ignore the desire for explicit authorization of middleboxes by the endpoints, and/or to only support all-or-nothing access models.

The Transport Layer Middlebox Security Protocol (TLMSP) [i.6] has recently been defined to address these shortcomings. At the same time, these ambitions of the TLMSP-design have led to a sophisticated protocol with a very broad range of configuration possibilities and there is a need to provide guidance for TLMSP implementers and users. The present document seeks to meet this need by providing a "handbook" of sorts, on how to use and configure TLMSP in different settings. Since TLMSP makes no claim of being the optimal solution in all middlebox scenarios, for completeness, the present document also discusses cases where TLMSP usage would require careful considerations and where, perhaps, another middlebox protocol might be preferred.

Introduction

TLMSP (ETSI TS 103 523-2 [i.6]) is a feature-rich protocol, and it could be unclear to some users how to configure it in different scenarios, i.e. when different application layer protocols are used, and when used for different services. The present document provides guidelines on the following topics:

1) How to map different parts (such as header fields, sub-parts of payload, etc.) of the application layer protocol onto different TLMSP contexts.

2) How to assign access rights (read/write/delete) to those contexts.

3) Recommendations on usage of deletions and insertions by authorized middleboxes.

4) Suitable format and content of TLMSP audit containers.

5) How to (in more specific detail) apply dynamic discovery of middleboxes.

In particular, items 2 to 5 depend not only on the application protocol, but also on specifics of the service running over that protocol and the business environment such as the type/functionality of middleboxes present, security threats, relevant security policies, etc.

The application layer protocols in scope are HTTP and protocols often used over HTTP, like XML, HTML, JSON, and YAML (including usage of OpenAPI™).

There are eight different types of services described in the same number of use-cases:

- audits in data centers;
- an enterprise firewall;
- content caching;
- load balancing;
- industrial automation;
- control systems;
- two telecom-oriented use-cases (roaming exchanges and service communication proxy); and
- finally a lawful interception use case.

1 Scope

The present document provides guidelines on how to use ETSI TS 103 523-2 [i.6] (TLMSP) in different scenarios; for example, when different application layer protocols are used, or when TLMSP is used for different services. It provides guidelines on the following topics:

- How to map different parts of the application layer protocol onto different TLMSP contexts.
- Which middlebox functionality that is relevant and how to assign access rights to allow the corresponding middlebox operations.
- Suitable trust model(s).
- Use of the different optional TLMSP features.
- Implementation and usability aspects.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] Joseph R., Biden Jr: "Executive Order on Improving the Nation's Cybersecurity", The White House, May 12, 2021.

[i.2] Virgil Gligor, Adrian Perrig and David Basin, "Determining an Economic Value of High Assurance for Commodity Software Security", 2023.

[i.3] IETF draft-rhrd-tls-tls13-visibility-01: "TLS 1.3 Option for Negotiation of Visibility in the Datacenter", R. Housley.

[i.4] NIST Special Publication 800-207: "Zero Trust Architecture", Scott Rose, Oliver Borchert, Stu Mitchell and Sean Connelly, August 2020.

[i.5] Cybersecurity and Infrastructure Security Agency: "Zero Trust Maturity Model", version 2.0, April 2023.

[i.6] ETSI TS 103 523-2 (V1.1.1): "CYBER; Middlebox Security Protocol; Part 2: Transport layer MSP, profile for fine grained access control".

[i.7] ETSI TS 103 523-3 (V1.3.1): "CYBER; Middlebox Security Protocol; Part 3: Enterprise Transport Security".

[i.8] IETF RFC 2616: "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing".

[i.9] IETF RFC 7616: "HTTP Digest Access Authentication".

[i.10] IETF RFC 4566: "SDP: Session Description Protocol".

[i.11] IETF RFC 4568: "Session Description Protocol (SDP) Security Descriptions for Media Streams".

[i.12] IETF RFC 3261: "SIP: Session Initiation Protocol".

[i.13] "The OAuth 2.0 Authorization Framework".

[i.14] OpenAPI™ Initiative (OAI).

[i.15] ISO/IEC 7498-1:1994: "Information technology -- Open Systems Interconnection -- Basic Reference Model".

[i.16] ETSI TS 123 501: "5G; System architecture for the 5G System (5GS) (3GPP TS 23.501)".

[i.17] ETSI TS 133 501: "5G; Security architecture and procedures for 5G System (3GPP TS 33.501)".

[i.18] ETSI TS 133 535: "5G; Authentication and Key Management for Applications (AKMA) based on 3GPP credentials in the 5G System (5GS) (3GPP TS 33.535)".

[i.19] GSMA IR.80: "Technical Architecture Alternatives for Open Connectivity Roaming Hubbing Model", version 3.0, , 2021.

[i.20] GSMA NG.132: "Report 5G Mobile Roaming Revisited (5GMRR) Phase 1", version 3.0, 2022.

[i.21] 3GPP Tdoc S3-234467: "LS to 3GPP re Monitoring of Encrypted 5GS Signalling Traffic".

[i.22] TLMSP OpenSSL.

[i.23] ISA/IEC 62443-1-1:2019: "Industrial communication networks -- Network and system security -- Part 1-1".

[i.24] ISA/IEC 62443-3-2:2020: "Security for industrial automation and control systems -- Part 3-2: Security risk assessment and system design".

[i.25] ISA/IEC 62443-4-2:2020: "Security for industrial automation and control systems -- Part 4-2: Technical security requirements for IACS components".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in ETSI TS 103 523-2 [i.6] apply:

3.2 Symbols

For the purposes of the present document, the symbols given in ETSI TS 103 523-2 [i.6] and the following apply:

`ctxt_id`	context identifier
`dm`	deleter MAC
`hbh_id`	hop-by-hop identifier
`wm`	writer MAC
*A	the element A repeated zero to any number of times
[A]	A is an optional element
A/B	precisely one of the elements A or B
CRLF	carriage return followed by line feed, i.e. "\r\n"
SP	space character
HTAP	tab character, '\t'
OWS	Optional White Space, i.e. * (SP | HTAB)

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI TS 103 523-2 [i.6] and the following apply:

AF	Application Function
A-KID	AKMA Key Identifier
AKMA	Authentication and Key Management for Applications
API	Application Programming Interface
ATP	Advanced Threat Protection
CA	Certificate Authority
CCA	Client Credentials Assertion
CDM	Continuous Diagnostic and Mitigation
CSP	Communication Service Provider
hCSP	home CSP
ETS	Enterprise Transport Security
GSMA	GSM Association
HTML	HyperText Markup Language
IPX	Internetwork Protocol eXchange
JOSE	JavaScript Object Signing and Encryption
JSON	JavaScript Object Notation
JWS	JSON Web Signatures
LEA	Law Enforcement Agency
LI	Lawful Intercept
MAC	Message Authentication Code
MNO	Mobile Network Operator
OSI	Open Systems Interconnection
OTT	Over-The-Top
PRINS	Protocol for N32 INterconnect Security
RH	Roaming Hub
RVAS	Roaming Value Added Service
SEPP	Security Edge Protection Proxy
SIP	Session Initiation Protocol
TLMSP	Transport Layer Middlebox Security Protocol
TLS	Transport Layer Security
UPF	User Plane Function
vCSP	visited CSP
ZT	Zero Trust
ZTMM	Zero Trust Maturity Model

4 Document Overview and Reading Guidelines

The present document is organized by first providing, in clause 5, a summary overview of the TLMSP protocol [i.6] and its features on a level sufficient for being able to follow the subsequent analysis. Then follows the use case analysis.

To that end, there are two natural approaches to cut the space of different use cases:

- it could be done based on non-technical characteristics such as the business environment (finance, utilities, telecom, etc.); or
- it could be based on more strictly technical aspects, such as looking at technical characteristics of the protocol which is to be transported over TLMSP (e.g. whether it is HTTP or something else).

In the present document, a choice has been made to perform the analysis based on both approaches, as presented in clauses 6 and 7. This choice was made since it is believed that both views have their own merit, and different audiences might prefer one approach over the other. While there are dependences and cross-references between these two parts, it has been a goal to make the respective clause stand on its own feet so that readers with a main focus on the business environment can focus on the presentation in clause 6 and readers with technical/security expertise can focus on clause 7.

In clause 8, user interaction and other implementation details of the TLMSP are discussed. This comprises both a management interface aspect, i.e. the glue between the application layer and TLMSP configuration and also how to interact with the user (e.g. via a browser) to allow user control and user notifications of what is happening inside the TLMSP layer. Clause 8 also discusses the integration of TLMSP into the OSI model. Clause 8 further presents a possible set of Application Programming Interfaces (APIs) for TLMSP, based on the open source distribution [i.21] with finer detail in annex A. Clauses 8 is thus mainly of technical nature.

Clause 9 discusses what role TLMSP can meet in implementing a Zero Trust architecture. Summary and conclusions are finally presented in clause 10.

5 TLMSP Overview

5.1 Introduction and TLMSP concepts

This clause provides a high level overview of TLMSP to facilitate understanding of the subsequent discussion without need to digest the whole TLMSP TS. Initially, only the basic features are discussed, and the more advanced features are discussed in clause 5.3.

Like TLS 1.2, TLMSP consists of a set of sub-protocols: Handshake, ChangeCipherSpec, Alert, and Application which are all carried over a common Record protocol. The Record protocol can have two different formats, of which the format for the Application and Alert protocols are the most important.

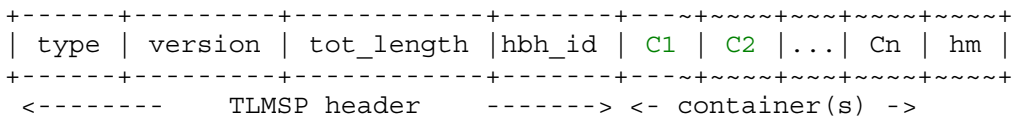


Figure 1: TLMSP record format (as used by Application and Alert protocols)

In Figure 1, C1, C2, …, Cn represents so-called containers. The field hbh_id (hop-by-hop ID) is used for enabling multiplexing of several TLMSP sessions on the same TCP transport connection.

A main feature is the usage of *contexts* and the associated *containers*. A context is a part of the application layer data that is to be protected by a certain policy, relative to the middleboxes. For instance, header (or metadata) information could be associated with allowing middleboxes to read the content, but not to change it, while payloads might be allowed to be modified by the middleboxes. Each middlebox can be granted its own access right (read/write/delete), independently for each context and independently for each middlebox. The exception is context 0 which is associated with the handshake itself and to which all entities have full access.

EXAMPLE: Middlebox M1 could have read access to context 1 but no access whatsoever to context 2, while middlebox M2 has write access to context 1 and delete access to context 2.

The container format is shown in Figure 2.

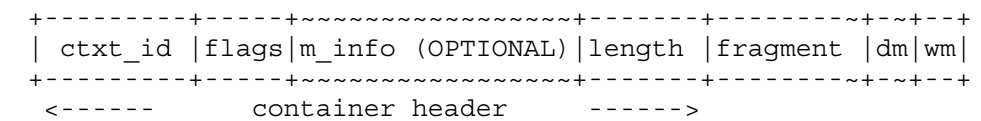


Figure 2: TLMSP container format

Application layer data associated with a specific context is carried in a container with the corresponding context ID in the header. The field `dm` is the deleter-MAC and `wm` is the writer-MAC. There is also a reader MAC, included as part of the `fragment`. The MACs are discussed in clause 5.3.

5.2 TLMSP Handshake

TLMSP is initiated by the client sending a standard TLS-formatted ClientHello, but with a special TLMSP extension, indicating to the server that the client is willing (or suggesting) to use TLMSP. This extension also contains an initial set of middleboxes, whose service the client is accepting or seeking. The client also indicates which access rights (read/write/delete) it wants to give each middlebox. Thus, it is assumed that the client has acquired knowledge on the available middleboxes by some out-of-band configuration. This message is forwarded on hop-by-hop TCP-connections between the topologically adjacent middleboxes, toward the server.

The middleboxes however remain silent during this process and only do forwarding, since it is not yet known if the server accepts (or even supports) TLMSP usage.

When the server receives the ClientHello, and if it supports TLMSP, it parses the TLMSP extension, decides if to accept the middleboxes' presence and access rights, etc. If so, the server responds with a ServerHello, also containing a TLMSP extension, and typically (see clause 5.3 for exceptions) following the path in reverse order, back to the client. At this point all parties (middleboxes and client) can determine the server's willingness to use TLMSP so at this point, TLMSP-specific handshake signalling can start.

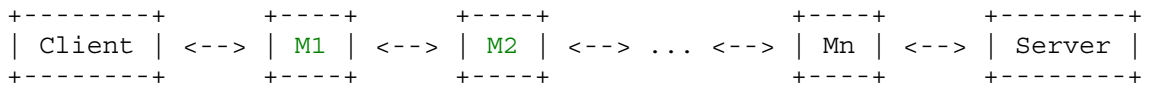


Figure 3: The TLMSP network architecture with client, server and middleboxes M1, M2, …

All middleboxes are fully visible and authenticate themselves to the endpoints (and to each other).

The most important part of the TLMSP-specific signalling are the messages associated with KeyMaterialContribution (directed from the endpoints to the middleboxes) and the KeyConfirmation (directed from the middleboxes to the endpoints). For each context, and each middlebox, the client and server each contributes a "half" of a key, that the middleboxes combine into the complete key. This makes it impossible for a middlebox to gain access to a context unless both client and server contributes their respective key-halves. Through the KeyConfirmation messages, the endpoint conversely gain assurance that the other endpoint has not on purpose refused to a give a certain middlebox the desired access right.

The handshake ends by verifying the integrity/authenticity of the handshake and enabling security for the TLMSP security layer, after which all records are protected by the chosen ciphersuite (activated by ChangeCipherSpec).

5.3 Implementing Access Control

Read-access is the simplest to implement and is done by simply granting a middlebox access only to keys for a context to which it is authorized to read. Separating write- (which includes insert) and delete-access is more involved since there is nothing that can prevent a malicious middlebox from attempting to e.g. modify or insert data associated with a particular context. This access control is instead implemented by adding Message Authentication Codes (MACs) to each container, using different MAC-keys to which conditional access can be granted. For example, a middlebox with only read access only has access to the key used to compute a so called reader-MAC, while a middlebox with write access has access also to the key for the so called writer-MAC. This means that unauthorized attempts to write/modify/insert or delete can be detected at the destination endpoint.

There is however one additional concern: since the endpoints generate the KeyMaterialContribution, this implies that an endpoint has full access to all contexts and could potentially bypass desired functionality of a middlebox.

EXAMPLE: There is a malicious server, sending malware which are removed by middlebox M1. If the server is able to access the connection between M1 and the client, the server could however re-insert the malware after it has passed M1, by adding authentic MACs to the packets.

To address this issue, there are also so called hop-by-hop-MACs, recomputed on each hop between middleboxes. This MAC is generated with a key that is unknown to any entity except the adjacent middleboxes.

5.4 Advanced Features

5.4.1 Middlebox Addition and Discovery

When the server receives the initial ClientHello, comprising the client's proposed middleboxes, it could be the case that the server has additional middleboxes it would like to add. To this end, the server's ServerHello extension could contain additional middleboxes which are up to the client to accept or not. In fact, as the ClientHello is forwarded to the server, the middleboxes themselves might want to add additional middleboxes. Such middleboxes are included in a way so that the server can differentiate the client's original set of middleboxes and the ones "discovered" during the handshake. Again, it is up to client and server whether to accept such middleboxes.

5.4.2 Audits

A middlebox which performs modifications, insertions or deletions could have a desire to communicate to the receiving endpoint which operation it has performed and why. To this end, authorized middleboxes can insert special audit containers, associated with a specific context. The format of the audit containers is not specified, and could in simple cases be human-readable information that could be displayed at the endpoint.

5.4.3 Sequence Number Handling

Handling of sequence numbers is much more complex than in TLS. This is due to the ability of TLMSP middleboxes to delete and/or insert containers, which alters the sequence number continuity. For this reason, and to prevent replay attacks, the TLMSP sequence number consists of an array of sequence numbers: one for each context, and one global sequence number, counting the total number of messages that has been passed. Further, there is one such array for each entity, "upstream" from the current receiver.

5.4.4 Middlebox Leave

It could happen that a middlebox becomes overloaded by serving too many connections. When this happens, the middlebox can request to leave the active session, i.e. stop performing the services it offers. However, the middlebox still remains on the path, but only performs forwarding of the messages.

5.4.5 TLS fallback and TLMSP Proxying

If the server does not support TLMSP, there is a built-in possibility for the last middlebox (closest to the server) to step in to terminate TLMSP, and to initiate a TLS-session on the hop between itself and the server.

5.4.6 Server Discovery

Although not described in the TLMSP specification, the middlebox discovery functionality as described in clause 5.4.1 can also be used to provide server discovery. The client would initially leave the server-entry in the middlebox list of the ClientHello empty. A middlebox (most likely the topologically last middlebox in the list) would perform server-selection by filling in the server-entry and forwarding the ClientHello to the selected server. The client would still have the option to accept or reject the selected server.

Server discovery can be useful for load balancing and similar use cases.

5.5 Cryptography

The cryptographic design of TLMSP follows best practices by supporting modern cipher suites such as AES-GCM, and providing perfect forward secrecy via ephemeral Diffie-Hellman.

6 Use Cases

6.1 Introduction

This clause describes a number of use cases (or "scenarios") in which it is likely to find relevant usage of TLMSP. These use cases are here characterized mainly in terms of the business case in which TLMSP is used, for example within cloud services, in telecom/mobile network setting, enterprise access, and so on.

Each use case description follows a common template describing:

- A use case summary description, comprising:
  - Environment: a short, high level textual description of the business environment, in particular the type of application (or sets of applications) deployed. This also includes assets and user subjects.
  - Threats: description of threats, focusing on those of particular relevance for the choice between TLMSP and hop-by-hop TLS.
  - Trust model: Which entities can be trusted? Which assumptions can be made? (Employees, TCB, etc.).
  - Security objectives: describes organizational policies and regulatory requirements that are likely to be relevant for the use case.
- Use case technical characteristics:
  - End-point types: describes the type of clients/servers are involved, for example whether clients are special purpose IoT devices, mobile phones or general purpose PCs.
  - Protocols: description of higher layer protocols that are to be transported and protected by TLMSP.
- TLMSP mapping, describing how the aspects above could likely be mapped to TLMSP features, including:
  - Middlebox functions: desired functionality to be provided by the middleboxes.
  - Middlebox access: the access rights are needed to perform the above functions.
  - Middlebox separation: whether there is need to separate duties of different middleboxes.
  - TLMSP contexts: classification of different parts of the application/protocols used (sensitivity, etc.).
  - Applicability and usage of optional TLMSP features:
    - Audit containers.
    - Dynamic middlebox discovery.
    - Middlebox leave.
    - TLS fallback.
    - etc.
- Discussion: an analysis of how well TLMSP addresses the use case, open issues for further study etc. As could be expected, TLMSP brings advantages in many cases, but it is also clear that there are use cases that might be equally (or perhaps even better) handled by other approaches. But instead of just omitting these use cases, it was felt to be of interest to still include them, giving the reader some insight also on when TLMSP usage needs to be more carefully analysed and compared to alternatives such as ETSI TS 103 523-3 [i.7] for example.

6.2 Audits in data centers

6.2.1 Use case description

6.2.1.1 Background

Some business sectors such as finance are subject to regulatory audit requirements. With TLS 1.2, it was straightforward to provide independent inspection of TLS sessions by delegating access to static keys to a trusted third party. TLS 1.3 removed static key options and although a draft [i.3] proposed a mechanism for exporting the (secret) ephemeral key to trusted third parties, this work was not adopted by the TLS working group in the IETF. ETSI TS 103 523-3 [i.7] (ETS) does provide a mechanism for exporting semi-static keys which will allow inspection of TLS sessions by a trusted third party even when they are not present during the establishment of the TLS session. However, ETS offers a lower level of forward security compared to TLS 1.3.

The remainder of the present clause discusses and analyses how TLMSP could be deployed in a data centre for auditing purposes; i.e. to allow TLMSP middleboxes operated by a trusted third party to audit transactions in such environments.

NOTE: Even though many countries have requirements on the audit of financial transactions, this can sometimes be performed at the endpoints themselves. This clearly gives limited assurance if the endpoints are compromised.

6.2.1.2 Security objectives, threats and trust model

The main security objective is obviously to ensure that only middleboxes operated by a trusted party can gain access which is a common baseline with security achieved by [i.3] and [i.7]. The main reason for using such 3rd party audit is to gain extra protection, in case the endpoints are compromised, e.g. when they are in control of a malicious party. However, since the security is here based on TLMSP, it would be beneficial to leverage further features that, strictly speaking, are not necessary, but still seem advantageous. Specifically, both solutions [i.3] and [i.7] are "all-or-nothing" solutions: they give the trusted third party access to all information and they further break not only end-to-end confidentiality but also end-to-end authenticity, which is something that can avoided by using TLMSP.

6.2.2 Use case technical characteristics

The endpoints are considered to be servers in a data centre, one taking the role of TLMSP client, and the other the role of TLMSP server. No specific assumption is made concerning the application layer protocol running over TLSMP, it is only assumed that the middleboxes have full knowledge of the details of this protocol and are able to parse the application transactions, e.g. financial transactions in a bank or between banks.

6.2.3 TLMSP mapping

It is assumed that there is typically at most a single middlebox present on each TLMSP connection, e.g. operated by (or on behalf of) a " national regulatory authority", though there could be more than one middlebox in some situations (for example, using two independent audit functions for extra assurance). In the simplest case, two TLMSP contexts are defined, one containing the application message units whose content are subject to audit (to which the middleboxes have read access), and one for "everything else" (with no middlebox access).

EXAMPLE: Suppose one would like to check that prices of a certain share is always correctly reflected in a stock market, and that this verification is done by an independent auditor. In such case, the purpose of a transaction (e.g. "sell 1200 shares in Dunder-Mifflin Paper Co at $42 per share") is to be audited, but the identity of the seller (and/or buyer) could remain private.

It cannot be taken for granted that the endpoints know if their transactions are potentially subject to audit at the point in time when they initiate the TLMSP handshake. However, general service discovery and load balancing functions are likely present in the data centre and could be leveraged in combination with the dynamic middlebox discovery feature when establishing a transport connection to the other endpoint. This would be necessary since unlike ETS, the middlebox needs to take active part in the handshake. However, there are also reasons why one might want middleboxes present in all sessions, see the discussion in the next clause.

Similarly, if a middlebox audit function has been present to audit one or more transactions between the endpoints, the middlebox might use the middlebox leave protocol when it no longer desires to audit.

6.2.4 Discussion

As seen, TLMSP offers a minimal disclosure policy with respect to the auditing function and it maintains end-to-end authenticity of all transactions, two features that the solutions [i.3] and [i.7] strictly speaking lacks. At the same time, there are some disadvantages with TLMSP. It seems at first to offer no great advantage in the case of compromised endpoints because the endpoints know if they are being audited or not and could choose to behave properly only when audited. To avoid this, middlebox participation in all sessions could be required. Furthermore, if both endpoints are compromised, they could collude and carry out "illegal" transactions inside those TLMSP-contexts that are not exposed to the audit function, if such contexts are defined. The fact that the middleboxes also need to be present during the handshake appears to be a performance disadvantage, though it is probably not significant.

6.3 Telecom 1: Mobile roaming exchanges

6.3.1 Use case description

6.3.1.1 Background

6.3.1.1.1 Roaming

The two use cases in the present clause are somewhat complex, so a substantial background description is first needed. Two other telecom-oriented use cases are presented in clauses 6.4 and 6.9.

The success of mobile networks in many ways owe thanks to the possibility of roaming: the business really took off when it became possible to use a single mobile subscription with a single Mobile Network Operator (MNO) to make and receive calls in virtually any network in the world. More or less from 4G an onwards, it has become possible not only to use a single subscription, but also a single mobile phone since the 4G Long-Term Evolution (LTE) radio network standard is a global norm.

6.3.1.1.2 Technical details of roaming

6.3.1.1.2.1 Basic principles

To make roaming happen, it is necessary to have networks interconnecting mobile operators and to define standard protocols to use over these networks. There are actually two kinds of interconnections needed, one for the inter-operator exchange of signalling data (e.g. retrieving information pertaining to the user's subscription) and one for the actual user traffic (e.g. IP packets to the Internet or VoIP packets). The focus of this use-case example is entirely on the signalling data.

NOTE 1: The interface for the user traffic is not required if so called "local break-out" is used, where user packets are routed to the service directly from the serving network.

NOTE 2: The user traffic is often carried over UDP, in which case TLMSP does not currently apply. Moreover, there are no clear "middleboxes" defined in standard for the roaming architecture.

Today, the signalling messages are carried over networks that go by the name Internetwork Packet eXchanges, or IPX networks, and are operated by IPX providers. In principle, the protocols used over these exchanges are defined by 3GPP. However, as will be discussed in more detail below, the IPX networks also provide services to operators, and the governance over the "roaming ecosystem" (e.g. how to handle roaming contracts and trust, how to configure the IPX networks, etc.), is managed under the GSM Association (GSMA).

For 5G, the inter-MNO roaming architecture, in its simplest form, is described in Figure 4.

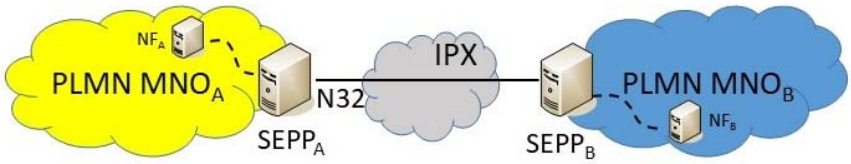


Figure 4: 5G roaming architecture

To allow subscribers of MNO_B to roam into MNO_A's Public Land Mobile Network (PLMN), and vice versa, some network function (NF_A) in the network of MNO_A typically needs to communicate with some Network Function (NF_B) at MNO_B. Here, N32 is the name for the technical reference point between the networks, whose details are defined in ETSI TS 133 501 [i.17]. The actual communication over N32 takes place over HTTP/2, and this communication is mediated by the Security Edge Protection Proxies (SEPPs). These are also defined by 3GPP and are responsible for policy enforcement, e.g. applying security on outbound N32-messages and verifying details of incoming messages before passing them on, as will be described in more detail later. The N32-messages are transferred using the services of the IPX interconnect provider.

NOTE: It is possible that the SEPP-functionality is outsourced, but in order to not over-complicate the example more than necessary, it is here assumed the MNOs themselves operate the SEPPs.

6.3.1.1.2.2 IPX services

In general, the situation is more complex because MNO_A and MNO_B might not be directly connected by the same IPX provider. Additionally, as mentioned above, there are strong business cases for providing services, so called Roaming Value Addes Services (RVAS) from inside the IPX-network(s).

EXAMPLE: An RVAS could provide services such as "welcome SMS" (when travelling across a border), message normalization (re-writing messages to comply with standard formats), message modification (e.g. removing data fields that are not supported by one of the MNOs), fraud detection, etc.

Therefore, a more realistic roaming architecture is provided in Figure 5, where the MNOs use different IPX provides inside the "IPX cloud" and where each of these IPXes also provide some service to the respective MNO.

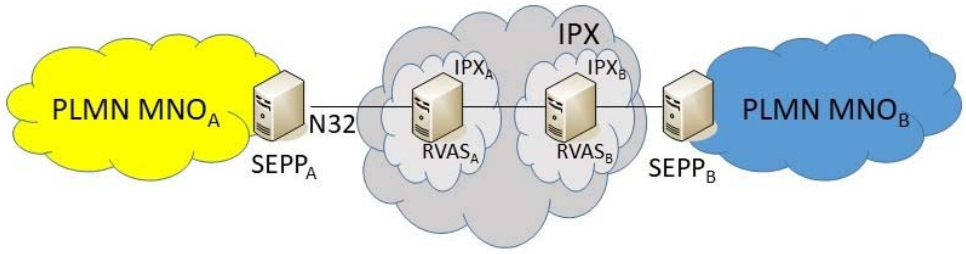


Figure 5: A more realistic roaming architecture

It is at this point obvious from the RVAS service examples above that there is a use case for middlebox functionality. There is on one hand a clearly visible need to protect the N32 messages (between the SEPPs) but if the RVASes are to be able to provide some kind of service, it appears necessary that they need at least to be able to read the contents of (parts of) encrypted messages and most likely also to be able to modify and/or even insert messages.

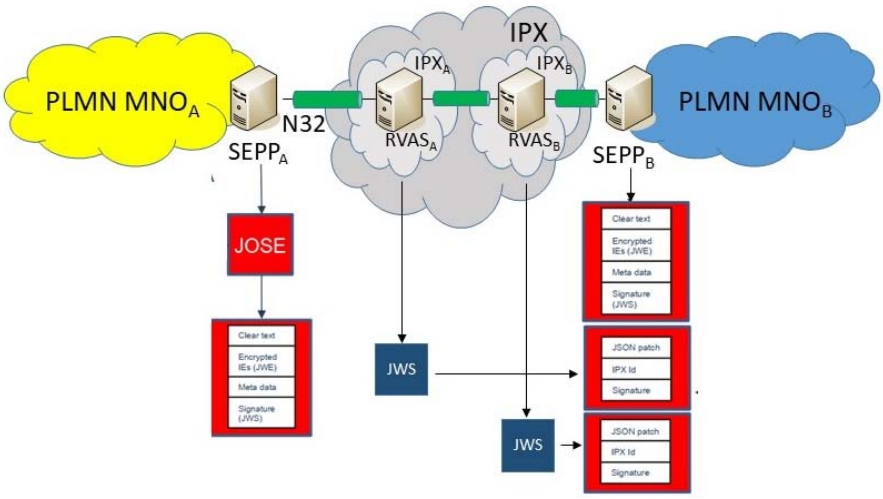
6.3.1.1.2.3 N32 security

Since the need for IPXes to be able to read and/or modify (parts of) N32 messages was known early in the 5G standardization work, 3GPP defined from the outset an N32-protection mechanism that is a kind of "middlebox protocol". The protocol is a combination of traditional TLS transport security applied to HTTP/2, on top of which a 3GPP-defined application layer protocol is deployed, the latter having the acronym PRINS (PRotocol for N32 Interconnect Security).

PRINS is defined in ETSI TS 133 501 [i.17] and can somewhat simplified be described with reference to Figure 6. The N32 messages from some function at MNO_A to some function at MNO_B are carried over HTTP/2, encoded as JavaScript Object Notation (JSON), and PRINS encrypts and signs these by using JSON Object Signing and Encryption (JOSE) at $SEPP_A$. However, encryption is only applied to those data fields of the JSON messages that do not require read (or write) access by the IPXes. This is achieved by splitting the JSON into two parts: `dataToIntegrityProtect` (and thus not to encrypt) and `dataToIntegrityProtectAndCipher`. After encrypting selected parts, the entire JSON message can be authenticated/integrity protected by applying a signature to the entire JSON message.

At this point the message is essentially ready to be forwarded over onto the IPX_A network, but that would allow anyone (including third parties) to read the unencrypted fields. Therefore, the JOSE-protected JSON message is sent over an encrypted TLS-connection to the neighbouring IPX_A.

$RVAS_A$ terminates the TLS protection and now has access to the JSON message and can read the data fields it is authorized to read and possibly also modify. (The IPX has been configured with a policy which allows it do distinguish those fields it can only read from those fields it can also modify.) In case $RVAS_A$ sees a need to modify/write a specific field, $RVAS_A$ does not actually carry out the modification, since that would invalidate the JOSE signature by the originating $SEPP_A$. Instead, $RVAS_A$ adds a so-called JSON patch, stating which modify/write operation *it would have liked to perform*, and signs this patch using JSON Web Signatures (JWS, with its own signing key). The $RVAS_A$ then forwards the message in a second TLS-connection toward $RVAS_B$, which acts in analogy with $RVAS_A$, possibly adding (and signing) a second JSON patch. Eventually, the message is received in a third TLS-connection at $SEPP_B$. $SEPP_B$ terminates TLS and then inspects the JSON message. First, it can verify the authenticity of the original message by validating the signature of $SEPP_A$ and can then decrypt applicable data fields. Next, it looks at the JSON patches by $RVAS_A$ and $RVAS_B$ (if any). Specifically, it checks if their signatures are valid and whether the suggested modifications are permitted (according to a pre-configured policy), and if so, the $SEPP_B$ *applies* the patches, thereby modifying the message, before forwarding it to the destination inside the MNO_B network. Figure 6 summarizes the message processing and forwarding just described.



NOTE: The green "pipes" correspond to TLS connections.

Figure 6: N32-security with PRINS

Regarding key management, there is one major thing to observe: the end-points need to obtain the certificates of both RVASes in order to be able to verify their JWS signatures on the patches. This will become important later.

This is clearly a case where two "middleboxes" between the SEPP-endpoints can be identified and where the middleboxes (RVASes) need a mix of read/write access privileges which TLMSP can support. It is also similar to TLMSP since end-to-end authenticity is maintained. A difference is that the writes are actually not performed at the middleboxes, rather the middlebox performs what can be viewed as inserting audit containers, whose content are checked at the destination endpoint which then performs the actual wite operation accordingly.

6.3.2 Sub-usecase 1: Application of TLMSP to N32

6.3.2.1 Security objectives, threats and trust model

The objective is to meet the same security objectives as those defined by PRINS, i.e. that the RVASes can read and modify (or more precisely: propose how to modify) according to some security policy defining the readable/modifiable parts. The trust model covers the SEPPs and the RVASes in the IPX network. The trust is as discussed limited since the RVASes can read only certain fields and can only suggest modification but not perform them.

6.3.2.2 Use case technical characteristics

Here, the application of TLMSP in a setting where the endpoints are MNO SEPPs communicating via IPX with two RVASes is considered.

On the path between the two SEPPs, there are thus two middleboxes corresponding to the RVASes.

The application protocol to protect is HTTP/2. There is no real need to be concerned with whether or not JSON (or something different) is run over HTTP/2 since there is no need to use the PRINS protocol at all. However, for concreteness it is assumed that JSON is still used in order to be able to rely on the discussion on how to map JSON to TLMSP contexts, as discussed in clause 7.4.

6.3.2.3 TLMSP mapping

The TLMSP endpoints correspond to $SEPP_A$ and $SEPP_B$ of Figure 7. Between $SEPP_A$ and $SEPP_B$ there will in the most general case be two middleboxes corresponding to the two IPX RVAS. Next, three TLMSP contexts are defined:

1) $Context_1$, which gives read access to those parts of the messages that need to be read and/or modified by the two IPX RVAS.

2) $Context_2$, which gives write/insert privilege to the RVASes.

3) $Context_3$, to which only the SEPPs have access.

Assuming it is desirable to implement the same policy as in PRINS, i.e. that $RVAS_A$ can only "propose" modifications (which are checked and implemented at the receiving SEPP), this can easily be accomplished by arranging so that $Context_2$ is used to carry the proposed modifications (i.e. the "patches") carried as TLMSP audit containers. In terms of delegating access to the middleboxes, both RVAS obtains access as stated above: read access to $Context_1$, write access to $Context_2$ and no access to $Context_3$.

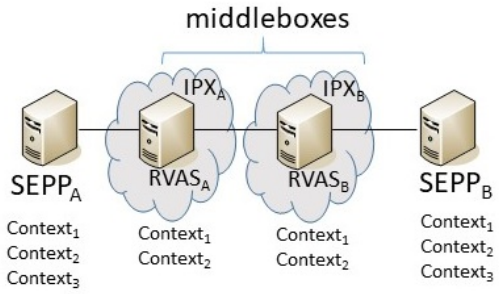


Figure 7: TLMSP set-up for N32 protection via two RVAS

6.3.2.4 Discussion

The most interesting feature used is related to delegating write/modify access directly to the RVASes, for those parts which they are authorized to modify, by using inserts (writes) in $Context_2$ rather than direct write/modify access of $Context_1$. The main reason is that with the second approach, it would no longer correspond to the same policy as enforced by PRINS in which the RVAS can only *propose* modifications; it would allow the RVAS to actually apply them directly causing issues to verify if only allowed changes were made.

While not described above, by using TLMSP it would be straight-forward to separate those parts that $RVAS_A$ can read/modify from those that $RVAS_B$ can read/modify. It would furthermore be possible to also separate parts that *both* RVASes can read/modify. This could be done by defining four more TLMSP contexts in analogy to the above, i.e.:

1) $Context_1$ is extended by adding $Context_{1A}$ and $Context_{1B}$, where $Context_1$ still covers those parts which both IPXes have access to and the two other contexts are used for those parts that only one of IPX $RVAS_A$ or $RVAS_B$, can read.

2) $Context_2$, is similarly extended by $Context_{2A}$ and $Context_{2B}$, handling write/insert privileges in analogy to above.

3) $Context_3$, to which only the SEPPs have access as above.

An advantage in using TLMSP compared to PRINS is the simplified certificate management. According to ETSI TS 133 501 [i.17], operators act as a certification authority for those RVASes that they have a direct business relationship with. In order to authorize N32-f message modifications, operators therefore sign a digital certificate for each of these RVASes and provide it to both the RVAS itself, as well as their roaming MNO partners to enable them to validate any modifications by this RVAS provider. If TLMSP is used, the (mutual) authorization of the RVASes by the MNOs can be established indirectly as a consequence of the fact that each MNO has provided key material associated with the relevant context(s) to the respective RVAS.

6.3.3 Sub-usecase 2: Application of TLMSP to N32 with roaming hub

6.3.3.1 Additional background

6.3.3.1.1 Roaming hubs

This use case is particularly interesting because it supports business cases where the business model has what some might consider "peculiar" impact on the trust model and how TLMSP needs to be aligned with this.

As presented in Figure 5, there is still a need for each pair of MNOs that want to provide roaming for their subscribers to sign pair-wise roaming business agreements, which could be problematic: there are some 800 MNOs world-wide, potentially leading to need for tens of thousands of such agreements. To avoid this, GSMA defines an entity called a Roaming Hub (RH). The basic idea behind the RH concept is that a MNO would in principle only need a single roaming agreement, with the RH. The RH then acts as mediator, and dynamically sets up roaming connections to any other MNO (that also has an agreement with that RH). This does not preclude that an MNO simultaneously has traditional roaming agreements with selected MNO, but the single agreement with the RH allows many roaming agreements to be handled in an on-demand, "bulk" fashion. Figure 8 depicts the RH architecture (omitting possible traditional roaming relations that could also be in place).

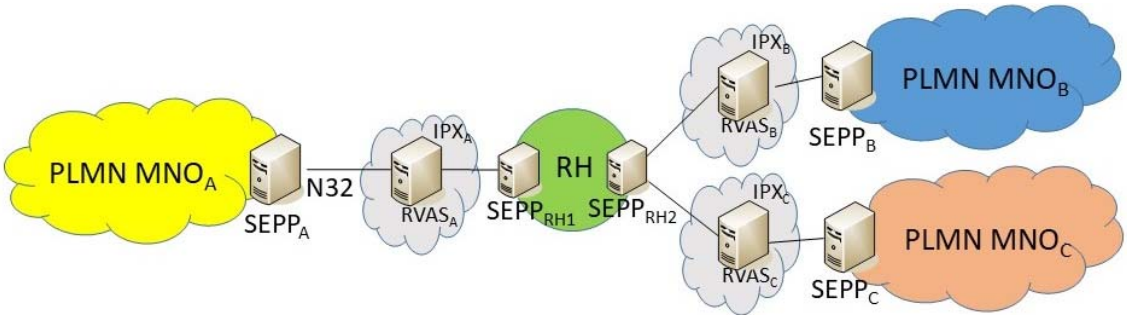


Figure 8: Roaming via roaming hub

Figure 8 shows one of two options in GSMA's architecture [i.19]. The other option is that the MNO-RH connections are direct, without reusing the IPX infrastructure. For the purpose of the present document, the specific choice of the IPX-variant above is made since it is more interesting from middlebox point of view. As seen in Figure 8, the RH also runs SEPP functionality. This is a requirement mainly for two reasons:

1) In order to enable the RH to take full responsibility for acting as a mediator, it is necessary to terminate security at the edge of the RH, otherwise the RH cannot see the full content of the signalling messages and consequently cannot assume liability in case something goes wrong, e.g. if some fraudulent behaviour occurs at the MNO end-points. So, the protection of a message from MNO_A to MNO_B needs to terminate at $SEPP_{RH1}$, and security is then re-applied at $SEPP_{RH2}$, as the message is forwarded.

2) Since the MNOs that use the RH service do not have any prior agreements, it is unlikely that a direct security association could be established between $SEPP_A$ and $SEPP_B$ anyway. For example, information required to mutually verify SEPP-certificates would need to be in place, which is most likely not the case.

6.3.3.1.2 Concerns about the N32 security solution

GSMA has voiced some concerns about the above described PRINS-based N32 security solution, with some more fundamental issues being introduced when considering roaming hubs.

The basic issues are:

- If there are no intermediaries (e.g. RVASes), skipping PRINS and using a direct TLS connection between SEPPs would be preferable. The choice between PRINS or direct TLS is a discrete choice to be settled between both 5G roaming partners [i.20].

- GSMA operators have reported implementation problems, and trouble-shooting and error diagnostics is not possible on the N32-protected traffic. Need to perform surveillance of traffic is also difficult for the same reason. Thus, adding a monitoring function that can decrypt and read traffic is needed [i.21]. Such a monitoring function is generically denoted as an MF.

For the second issue, GSMA proposes that the SEPP could send an additional "mirror copy" of the traffic to a monitoring function. However, this does not work if there is suspicion of malicious behaviour by the SEPP.

When roaming hubs are added to the picture, more serious concerns show up:

- If the RH is to take full responsibility (including e.g. financial liabilities) for handling roaming agreements, it simply requires full access to message content. Thus, the RH cannot be handled in the same way as the IPXes, having at best partial access.

- Even if RH faces no liability, it needs to be able to keep its business relations confidential.

EXAMPLE: It probably cannot disclose with whom it has business relations (for example, which other roaming hubs it cooperates with) by forwarding certificates on behalf of the endpoints.

This implies that when a RH mediates roaming agreements, it cannot at the same time mediate any end-to-end security association between the end-point MNOs. Rather, the NH needs to have two distinct security and trust relations, one with MNO_A and one with MNO_B (or a second roaming hub).

6.3.3.2 Security objectives, threats and trust model

The aim is to meet the same security objectives as those defined in clause 6.3.2.1, but between a MNO and the RH. Additionally, the objective is to allow a monitoring function having full read access, but no higher privilege. The trust model covers the SEPPs, the MF and the RVAS in the IPX network but where the SEPP is not assumed to be operating 100 % correctly (else there would be no need for the MF).

6.3.3.3 Use case technical characteristics

The application of TLMSP in a setting using roaming hubs is considered, which implies that the endpoints correspond to an operator SEPP and the SEPP of the RH.

As discussed in clause 6.3.3.1.1, the endpoints cannot be the two operator SEPPs since:

- A pre-existing trust relation between the MNOs cannot be assumed, which would contradict the relevance of the RH.

- A security-terminating SEPP at the RH is required in order for the RH to assume liability.

This does not preclude that TLMSP could not also be used between the second RH SEPP and the second MNO, but that usage would be handled similarly, mutatis mutandi and does not require a separate explanation. On the path between the two SEPPs, there are two middleboxes in the most general case: one corresponding to the traffic monitoring function of MNO_A motivated by the discussion in clause 6.3.3.1.2, and one corresponding to the intermediate IPX RVAS.

The application protocol and other details are identical to those of clause 6.3.2.2.

6.3.3.4 TLMSP mapping

The mapping is similar to that of clause 6.3.2.3 but with slightly different access right (caused by the need to allow MF_A to monitor all traffic). The TLMSP endpoints corresponds to $SEPP_A$ and $SEPP_{RH1}$ of Figure 9 (i.e. $SEPP_{RH1}$ takes the role of $SEPP_B$ of Figure 7). Between $SEPP_A$ and $SEPP_{RH1}$ there will in the most general case be two additional middleboxes: one corresponding to the traffic monitoring function of MNO_A (denoted MF_A below) and the IPX $RVAS_A$. Next, three TLMSP contexts are defined:

1) $Context_1$, which gives read access to those parts of the messages that need to be read and/or modified by IPX $RVAS_A$.

2) $Context_2$, which gives insert/write privilege to $RVAS_A$.

3) $Context_3$, which gives read access to all the remaining parts which are not covered by $Contex_1$ or $Context_2$. (This is intended to be used by the monitoring function MF_A.)

The first two contexts serve the same purpose in granting access to the RVAS as discussed in clause 6.3.2.3, i.e. $RVAS_A$ obtains access to $Context_1$ and $Context_2$. No access is granted them to $Context_3$. MF_A is assumed to have read access to all content originating at the $SEPP_A$, so it is assigned access to $Context_1$ and $Context_3$ (i.e. full read access, but nothing else). The MF does not need any access to $Context_2$, since no information is originating at $SEPP_A$ for this context. Figure 9 shows the set-up.

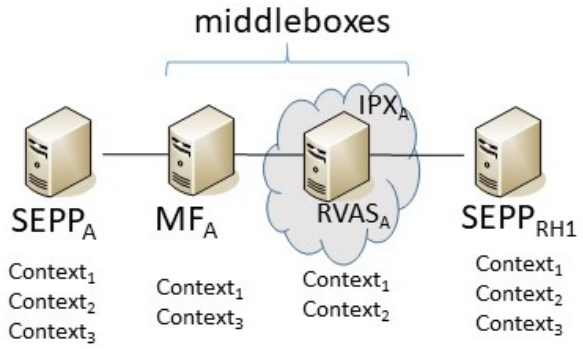


Figure 9: TLMSP set-up for N32 protection via roaming hub

6.3.3.5 Discussion

Comparing to the previous use case, there is here one other aspect entering the picture leading to the need for the third context with separate privileges. Recall that MF_A is supposed to have full read access, but *nothing beyond that*. If one were to define a context with write/modify privilege (intended for $RVAS_A$) one could not grant also MF_A access to that context. One would then need to duplicate the modifiable parts of the messages and send a copy of it inside a separate context to which MF_A *only has read access*. But as mentioned, that does not work in case the $SEPP_A$ is compromised since there is then no reason to trust that data copied is not faked as legitimate while the original (inside the other context) is "bad".

6.4 Telecom 2: Service Communication Proxy

6.4.1 Use case description

6.4.1.1 Service based architecture and communication proxies

The 3GPP-defined 5G network (ETSI TS 123 501 [i.16]) utilizes a Service Based Architecture (SBA). Different Network Functions (NFs) register themselves and the services they provide in a Network Repository Function (NRF). Another NF that requires usage of a specific type of function (a so called "consumer") can, via the NRF, discover which NFs that are available to provide the desired services (a "producer") and correspondingly select one appropriate instance NF. This is depicted in Figure 10.

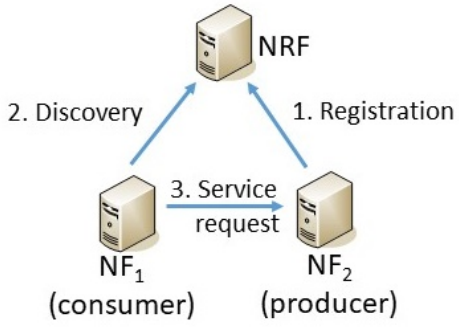


Figure 10: 5G SBA overview

The discovery can return a list of plural available producer instances, so an instance selection might need to be performed by the consumer.

The communication between consumer and producer uses HTTP/2 and is thus typically TLS protected. The producer can authorize the consumer by relying on (assumed) physical perimeter security of the network, TLS certificates, or, special authorization tokens issued by the NRF.

For reasons such as simplification of network topology, signalling aggregation, routing, load balancing, overload handling, message parameter harmonization, monitoring, policy enforcement, and interworking with legacy system, it could be beneficial to introduce an intermediary between the producer and consumer NFs. In the context of 5G, such an intermediary is called a Service Communication Proxy (SCP) and was introduced in the second 5G release (corresponding to 3GPP Release 16). Each NF can be associated with a so called SCP domain, defining the set of NFs that can communicate directly with a certain SCP, and this information is stored in the NRF. If and how to include an SCP between the end-point NFs is done mainly in two ways, as described in ETSI TS 123 501 [i.16]:

1) After discovery of producer NF (which includes SCP domain information) as provided by the NRF, the consumer NF could send its service request via a SCP and delegate to the SCP to select a specific producer NF instance.

2) The consumer NF could skip NRF-provided discovery entirely and instead send a service request directly to the SCP, which is then delegated *both* the task of producer NF discovery as well as instance selection (within the SCP domain).

In the second case above, there is in a sense two connections mediated by the SCP, the producer NF discovery which terminates in the NRF, and the actual signalling traffic, terminating at the producer NF. The two main cases are depicted in Figure 11.

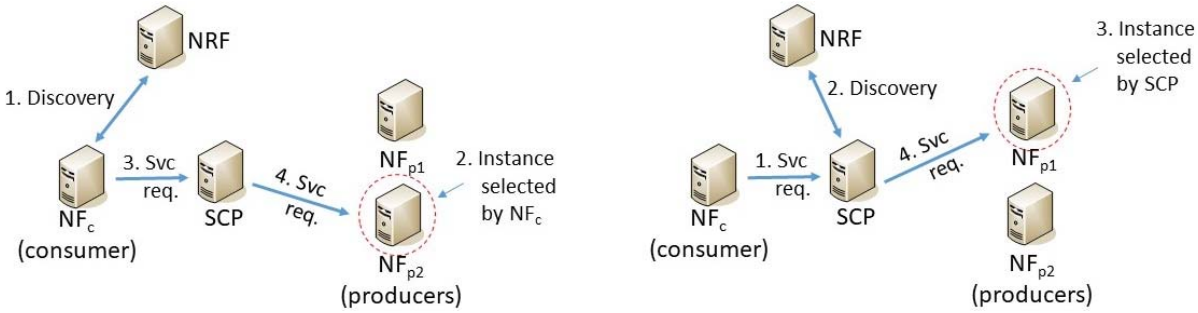


Figure 11: The two use cases of SCP

It is possible that the SCP is co-located with the NF. This case is however not discussed further.

6.4.1.2 Security objectives, threats and trust model

The following requirements are stated in ETSI TS 133 501 [i.17]:

1) The interface between the SCP and the NFs and between the SCPs needs to provide:

   a) mutual authentication; and

b) communication security (confidentiality and integrity).

2) If the signalling message (service/subscription request or notification message) from the sending NF does not include the network identity but the sending SCP can determine that identity, the sending SCP preferably includes it. This is done in roaming to allow another network/MNO to authorize messages when they are sent over N32 as described in clause 6.3.1.

All parties communicating through the SCP are assumed to trust the SCP to correctly handle the messages passing through it. This is crucial since ETSI TS 133 501 [i.17] by 1) relies on hop-by-hop TLS authentication/protection between the endpoint NFs and the intermediate SCP.

6.4.2 Use case technical characteristics

In non-roaming cases, SCP communication takes place between two NFs, via the SCP. It is also possible to use SCP in roaming as described in clause 6.3.1. In that case, the connection endpoints comprise an NF and the SEPP, with the CSP in between them. This later case is however not discussed further.

6.4.3 TLMSP mapping

Recall that the 5G SBA communication is using HTTP whose handling is described in clause 7.3 The 3GPP specifications for 5G also provide YAML protocol specifications, which is discussed further in clause 7.5. The endpoints consists of the consumer and producer NFs with the SCP corresponding to the middlebox. The SCP could either be a pre-configured middlebox of the consumer, or could be provided during producer NF discovery as described in clause 6.4.1.1.

If discovery is performed by the consumer NF, there is only one TLMSP session, carrying the signalling traffic between the consumer and producer NF. If the producer NF selection is done by the SCP, the consumer NF, when initiating TLMSP will not know the identity of the producer NF (corresponding to the TLMSP server). In this case, the consumer would use the dynamic server assignment feature discussed in clause 5.4.6, i.e. send a middlebox list with an empty server entry, and the dynamic discovery mechanisms of TLMSP would allow informing the consumer of which server/producer has been selected (by the SCP), by sending back a new TLMSP middlebox list with the server field filled in. The consumer NF would the re-initiate TLMSP with the specified producer. Which contexts are need is dependent on the services provided by the producer NF, but as can be seen from the example usages discussed in clause 6.4.1.1, the SCP might need any or all of privileges from the set {none, read, write/insert, delete}, thus implying up to four different contexts.

If both selection and discovery is delegated to the SCP, similar principle applies. However, as discussed, it is now also possible to view this as two separate TLMSP sessions via the SCP, one for the discovery phase (terminating at NRF), and one for the signalling payloads (terminating at the selected producer NF). In this case, the TLMSP session associated with the discovery phase would use two contexts: a $Context_1$ with only read privileges for the SCP and $Context_2$ with write/insert (and possibly delete) privilege for the SCP, into which the SCP injects the discovery messages. Unless some information about the discovery needs to be sent back to the consumer NF, the SCP omit forwarding $Context_2$, except possibly as TLMSP "delete notifications".

EXAMPLE: The consumer NF sends as special token in $Context_1$ (this could include the identity of the SCP). The SCP (who forwards this to NRF) also inserts a discovery message into $Context_2$. The NRF, when receiving the messages, will from the first token be able to determine that the consumer NF indeed has authorized the SCP to perform discovery, whereas the message in $Context_2$ serves to actually perform the discovery.

NOTE: The 5G Security Architecture in ETSI TS 133 501 [i.17] provides a similar mechanism , known as Client Credentials Assertion (CCA).

Figure 12 shows the case of SCP-delegated discovery, followed by utilization of the producer NF services. In Figure 12, TLMSP session 1 is used for the discovery delegation and session 2 for the actual SBA service. It is assumed the SCP services require access to $Context_2'$ (and possibly some other contexts of TLMSP session 2) but not to $Context_1'$.

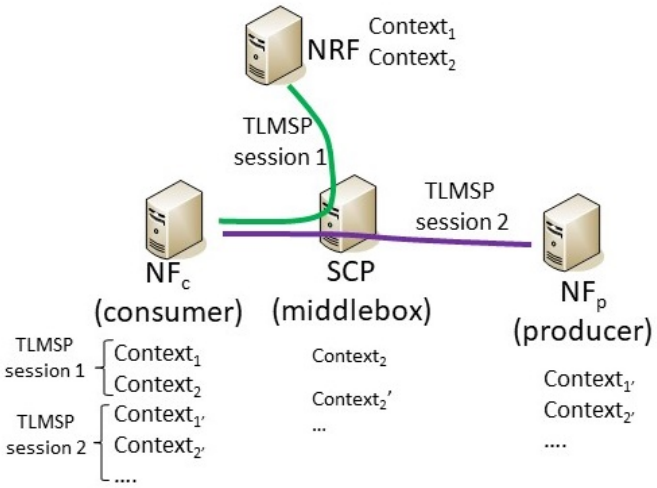


Figure 12: SCP-delegated discovery and SBA service utilization

6.4.4 Discussion

With TLMSP, end-to-end authentication can, unlike the currently defined hop-by-hop TLS-solution, be achieved even in the presence of SCPs.

An interesting option occurs in the case where discovery is delegated to the SCP by treating that as a separate TLMSP session. TLMSP could as discussed be leveraged to allow the NRF a more explicit verification that the consumer NF has actually delegated the discovery to the SCP. There is however an associated cost since an additional TLMSP handshake is needed.

While roaming was not discussed, it would (at least in principle, and without roaming hubs) be possible to use TLMSP fully end-to-end in the roaming case, where the communication would take place e.g. as:

$$NF_A - SCP_A - SEPP_A - RVAS_A - RVAS_B - SEPP_B - SCP_B - NF_B$$

i.e. with no less than six middleboxes.

6.5 Enterprise Firewall

6.5.1 Use case description

6.5.1.1 General

One feature of modern firewalls is the ability to terminate TLS traffic, which enables traffic inspection and also the ability to scan for viruses and to do Advanced Threat Protection (ATP). For the user, protection against cyber threats is vital. But termination of TLS has drawbacks, such as loss of secrecy and integrity of the user data. There is also no visibility of the server identity or certificate. Due to these drawbacks the firewall needs to be fully trusted.

The use of TLMSP will reduce these drawbacks by giving the end user control over what part of the data the firewall will have access to and if the access is limited to read-only access.

Note that this use case in essence is very similar to monitoring and control in "Industrial automation and controls systems" (see clause 6.8) that also deals with trying to combine the two conflicting forces of visibility and encryption.

6.5.1.2 Security objectives, threats and trust model

A desirable property is that even if the firewall is given access to different parts of traffic content, it is still possible to establish a mutual identity/authenticity relationship between client and server, but also to remain certain that information integrity is conserved in the data flow between client and server.

Further, the firewall's access to different parts of the payload data flow between endpoints remains under endpoint control. In other words, the firewall is not really trusted to perform actions outside that of scanning traffic with the purpose to block unintended traffic and to identify threats.

6.5.2 Use case technical characteristics

A technique used by firewalls, web proxies and similar appliances consist of intercepting, terminating, and re-encrypting the TLS traffic between the endpoints. This is accomplished by the firewall by serving as an intermediary that connects to the server instead of the actual client. Traffic going back to the client is encrypted based on a certificate generated by the firewall which means that the client should be able to trust that certificate, i.e. there should be a verifiable certificate chain back to the CA certificate installed in the client.

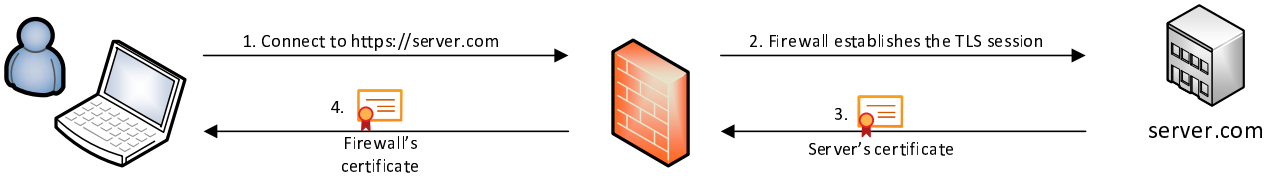


Figure 13: Flow of operations in TLS inspection

This flow is illustrated in Figure 13:

1) The client tries to connect to e.g. http://server.com.

2) The TLS session establishment is intercepted by the firewall that establishes another TLS session with the server.

3) The server uses its own certificate to authenticate with the server. Encryption is based on this certificate.

4) The firewall responds back to the client using its own certificate. The client should trust this certificate.

The firewall's certificate needs to be trusted by the client and that can be accomplished either by using a (by the firewall) self-signed certificate, or by letting the firewall have access to a certificate that is part of a certificate chain trusted by the client.

There are several disadvantages in this architecture:

- The firewall has full read/write access to all content this the traffic flow. I.e. the client should fully trust the firewall with all its content and cannot be certain nothing has been modified. In most cases the firewall does not need write access to provide its services.
- The client cannot be certain that it is really communicating with the server (server.com) in this case.
- The server does not know from which client the traffic originated. It can only see the firewall.
- The client can only be assured that the communication is encrypted to the firewall and needs to trust that the firewall correctly encrypts all communications between the firewall and the server.

The features of TLMSP could be leveraged to mitigate these disadvantages by allowing the client to authenticate both the server and the firewall and to provide a tool for content access control to the client.

6.5.3 TLMSP mapping

Depending on the scenario there could be almost any number of TLMSP endpoints involved but there are typically a minimum three: client, firewall and server. However, the firewall could involve more middleboxes for different services such as anti-malware detection, intrusion detection and parental control. Using TLMSP, the firewall can be given proper access rights depending on the service provided by the firewall (e.g. intrusion detection, malware scanning, parental control, etc.).

6.5.4 Discussion

As shown in this clause the usage of middleboxes and TLMSP would let firewalls and other types of network devices and services to have access to content but at the same time provide the end user with much better means to control this access. The firewall could of course block communications (assuming that it is in the path between the client and the server) in case the client does not allow access to its content. However, the decision to disclose content can now be taken knowingly by the end user.

6.6 Caching

6.6.1 Use case description

6.6.1.1 General

A cache is used for instance to reduce load to, or to reduce the latency to a service or content. There is more than one possible architecture for a cache-based solution. Some are however susceptible to cache poisoning.

EXAMPLE 1: The cache might be a separate instance of the service, closer to the client, which the client accesses first in an attempt to query some material. If the cache does not have the content, the client asks the main server to retrieve it instead. For other clients to benefit from that, the client need not to forget forwarding the content to the cache once it got its hands on it. Here cache poisoning is an issue, so is the common desire for client transparency.

EXAMPLE 2: It is the cache which makes the connection to the server when it does not have the requested resource. The cache then stores a copy if other users make the same request later.

6.6.1.2 Security objectives, threats and trust model

In the first example above, the cache trusts the client. A malicious client would be able to inject bogus records into the cache.

In the second example, the client trusts the cache.

6.6.2 Use case technical characteristics

A typical technical setting would be a mobile phone subscriber accessing content over mobile broadband, or an "edge cloud" solution where an enterprise desires low latency access between devices and a data centre.

6.6.3 TLMSP mapping

A possible TLMSP mapping ought to be of the following type: The cache acts as a middlebox, and, not considering any authentication process, has read access to all contexts (which contexts are needed is application dependent). A context, C_{aS}, reserved for cache-to-server communication where delete permission for the cache ought in general to be useful. Moreover, there ought to be a context, C_{CI}, for cache to client insertions.

In the C_{aS} context, the cache informs the server of the cache's own status, e.g. has the content been cached, and at which time. A context C_{cont} might optionally be used to handle content.

A typical flow for a deployment similar to example 2 above might be as shown in Figure 14.

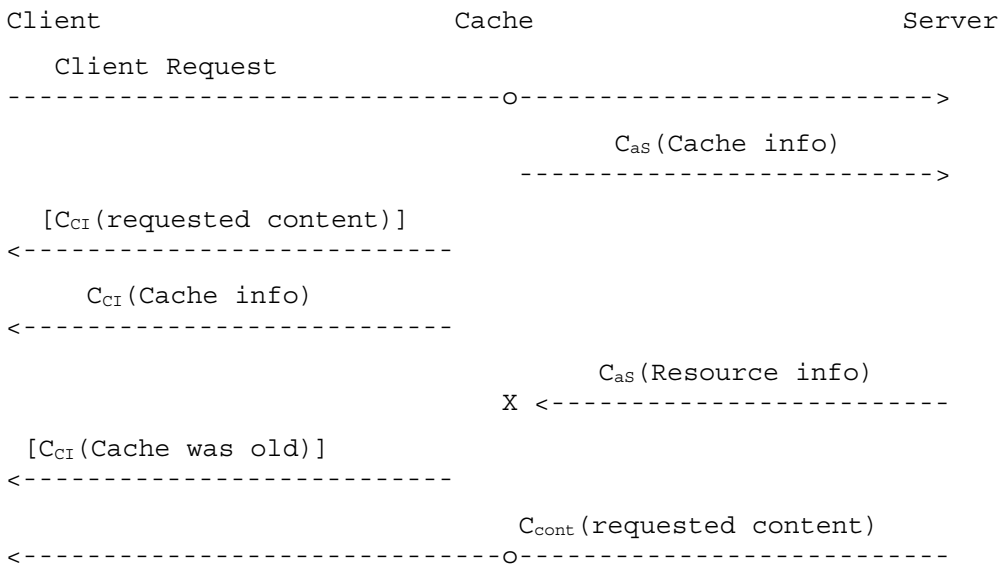


Figure 14: Flow via the cache. C( ) denotes data transferred as part of context C, whereas "X" and "o", denotes messages deleted and forwarded, respectively

6.6.4 Discussion

If latency is an issue, the proposed solution would not be optimal when adding the TLMSP handshake time to the total time. However, if the connection remains up for a number of requests, this will only be a problem for the first request. The use of TLMSP also limits the transparency toward the client.

On the other hand, with TLMSP, authentication is provided between all participating parties as well as integrity for the "control channels" in context C_{as} and C_{cl}. (It is otherwise quite common to let the cache have access to the server keys, providing transparency at the cost of strong authenticity.) At least the first user who requests certain content, gets end-to-end assurance of the authenticity (with TLS, the client does not get any assurance that the content is actually originating from the servers). This could be extended to clients who never sees the original server's content in full by combining the client's data fetches from the cache (inside a first TLMSP context) with allowing the server to provide the client with compact fingerprints (hash values) of original content (inside a separate context, with at most read-access granted to the cache).

6.7 Load balancing

6.7.1 Use case description

6.7.1.1 General

A load balancer typically distributes incoming requests/connections from a large number of clients to a set of servers. The servers register somehow to the load balancer. Clients are not able to connect to a server directly. In its simplest form the load balancer uses round robin to select which server to forward the incoming request to, and it does not inspect the traffic. The next step would be for the load balancer to poll the servers at regular intervals for their current health. That would then be done out of band.

6.7.1.2 Security objectives, threats and trust model

A desirable property is that even if the load balancer selects the server, it is still preferably possible to establish the mutual identity/authenticity between client and server. Further, the load balancer's access to different parts of the payload data flow between the end-points is preferably under end-point control. In other words, the load balancer is not really trusted to perform actions outside that of locating and assigning physical resources in the proper way.

6.7.2 Use case technical characteristics

A typical example would be clients accessing a service which at times can be heavily loaded so that distribution across several physical service instances becomes necessary.

6.7.3 TLMSP mapping

With TLMSP, a possible way of constructing a load balancer would be to have at least two contexts. One, used for the client-server end-to-end secured exchanges (which the load balancer has no access rights to), and another one called the load balancer status context, which the load balancer has read and delete rights to.

The clients sends a TLMSP ClientHello to the load balancer. Either the load balancer has only one set of servers to serve, or it looks into the server name indicator extension to decide which server to forward the client hello to. The choice would then be based on information of the servers which they have given to the load balancer. When the ServerHello reaches the client, it can see that the TLMSP extension has been updated with the server identity as discussed in clause 5.4.6.

When the handshake has been completed, the client and server can exchange their data as needed. However, the load balancer status context is used by the server to report to the load balancer of its current load and other useful information for the load balancer to make the next decisions on. The load balancer would then delete that information, because it is internal to the service, so that the client does not receive it. In its simplest form the load balancer would have no access to the data being sent between the client and the server. However, that is possible to change when needed.

6.7.4 Discussion

The objective of end-to-end assurance of the server/client identities can be met, as well as controlling the load balancer's access to exchanged information. None of these would be possible with standard TLS back-to-back approaches.

6.8 Industrial automation and control systems

6.8.1 Use case description

6.8.1.1 Introduction to ISA/IEC 62443 series of standards

ISA/IEC 62443 is a series of international standards developed jointly by the International Electrotechnical Commission (IEC) and the International Society of Automation (ISA). These standards focus on the cybersecurity of Industrial Automation and Control Systems (IACS), commonly referred to as Operational Technology (OT) systems. The ISA/IEC 62443 series of standards provides guidance and best practices to help organizations secure their IACS environments against cyber threats.

The standards are used by operators, system designers and vendors within OT and consists of multiple parts, each addressing different aspects of industrial cybersecurity.

The system level (level 3) specifies a structured risk-based method to deduce a zoned architecture and level 4 provides more detailed requirements on IACS components. Information flows between the zones is called conduits and the flows shall be monitored and controlled by network devices at zone boundaries, see security requirement NDR 5.2 in ISA/IEC 62443-4-2 [i.25].

Although the requirement does not state the level of monitoring and control required it is a reasonable assumption that a successful attack against a device within a specific zone can be exploited to break the compartmentalization. It has been shown by for example Vedere Labs in OT:ICEFALL how easily OT devices can be exploited over the network with arbitrary code execution as a result. In conclusion, allowing arbitrary traffic (i.e. encrypted unmonitored traffic) to these devices does not comply with the intention of the requirement.

6.8.1.2 Zones and conduits according to ISA/IEC 62443 series of standards

The concept of partitioning industrial automation and control systems into zones and conduits is introduced in ISA/IEC 62443-1-1 [i.23] and from a risk perspective further developed and described in ISA/IEC 62443-3-2 [i.24]. Zones and conduits are deduced using a risk-based approach and assigned a security level. These conduits connect two or more zones, and this inter-zone communication is controlled and monitored according to the assigned security level.

This segmented approach and traffic monitoring between zones gives rise to the need to create observability in the traffic flows.

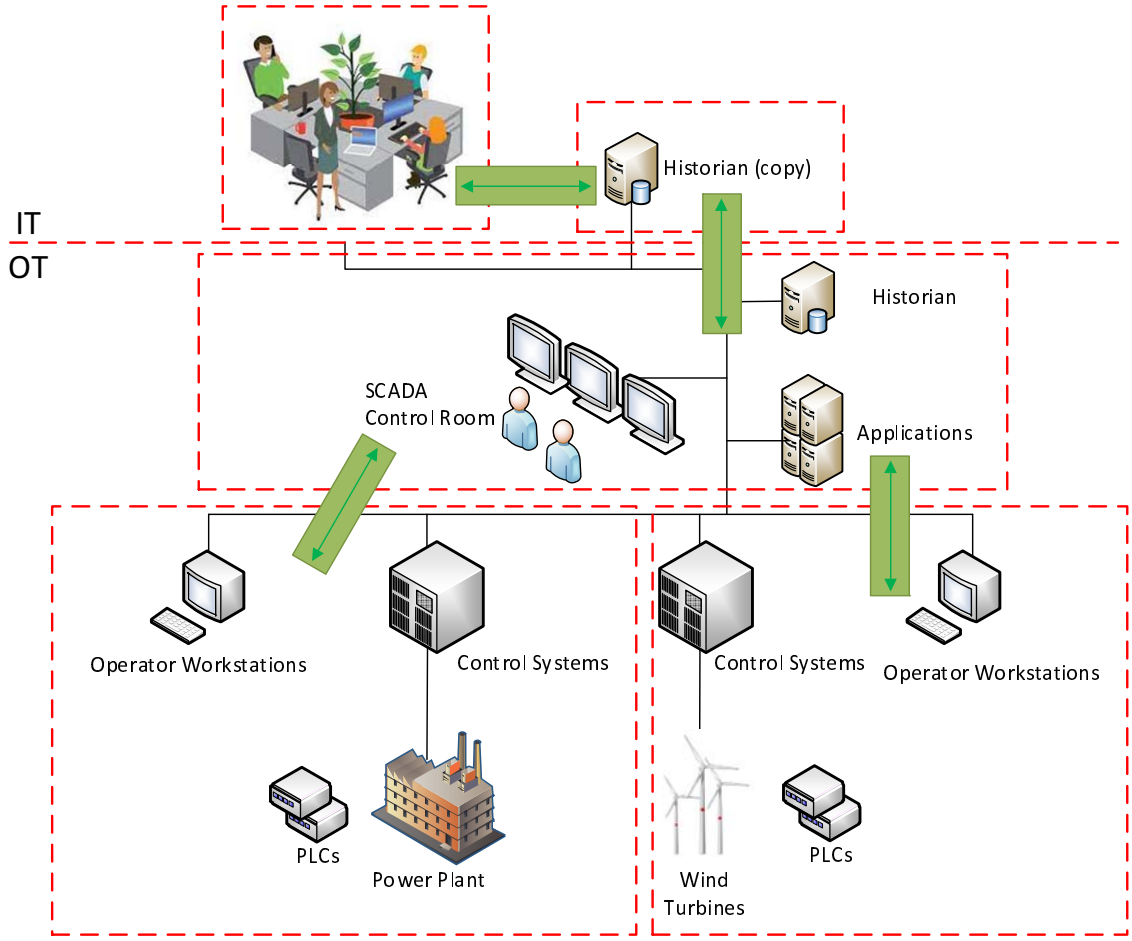


Figure 15: Illustration of the zones and conduits concepts

Figure 15 illustrates the application of zones and conduits according to ISA/IEC 62443 and the need for traffic monitoring emerges in the conduits between zones. However, the introduction of encryption between systems in these types of environments complicates this, and in some cases, renders it nearly impossible to become ISA/IEC 62443-compliant. For example, this is the case when introducing encryption between the Programmable Logic Controllers (PLCs) and the central SCADA system, especially when the PLC is in a different zone than the SCADA system, which is a very common situation given the fact that SCADA system controls a multitude of PLCs and underlying systems that could be spread over different facilities and even different factories.

To be compliant with ISA/IEC 62443 series of standards, the traffic to the PLC needs to be monitored and scrutinized when leaving and entering the SCADA zone.

6.8.1.3 Security objectives, threats and trust model

The security objectives consist of the following:

- Protect the PLCs from threat agents trying to disrupt the process by sending illegit (and probably unauthenticated) protocol messages or commands to the PLCs.
- Protect the PLCs from threat agents trying to disrupt the process by sending legit (often unencrypted and unauthenticated) commands with improper values to the PLC. This is applicable for all protocols without proper authentication, which is unfortunately not uncommon among older protocols, e.g. MODBUS.
- Protect the SCADA system from attackers using rogue PLCs and lateral movement attempting to reach the SCADA system.
- Prevent untrusted monitoring middle-boxes from tampering with the information flow but still allowing them to read/monitoring the traffic. This removes the attack vector of middleboxes tampering with measurement data or control commands.

6.8.2 Use case technical characteristics

The endpoints are considered to be servers, workstations and various IoT-devices, taking the role of either TLMSP client or the role of TLMSP server. No specific assumption is made concerning the application layer protocol running over TLSMP, it is only assumed that the middleboxes have full knowledge of the details of this protocol and can parse the application transactions.

6.8.3 TLMSP mapping

A number of PLCs are logically linked over a network to the SCADA system and continuously delivers measurement data into SCADA. The SCADA system can also send control commands to the PLCs. Common protocols involved are MODBUS, OPC-UA/DA, MMS, and MQTT.

In order to add encryption but still be able to monitor the traffic between security zones, TLMSP middle-boxes are introduced at the plant floor level. The TLMSP tunnel is then terminated at the border of the Main facility zone but can due to the properties of TLMSP be decrypted and monitored by middleboxes (see Figure 16) as the traffic flow traverses intermediate zones. For example, this allows inspection of the traffic before it enters the plant facility and protects PLCs from attackers that have gained access to either the plant zone or the SCADA zone.

The TLMSP client is a device on the plant floor (except when using a TLMSP Gateway according to clause 6.8.4) and the TLMSP server is either the SCADA server or the firewall protecting the SCADA server. The middlebox would typically be firewalls located at the zone boundaries (see Figure 16).

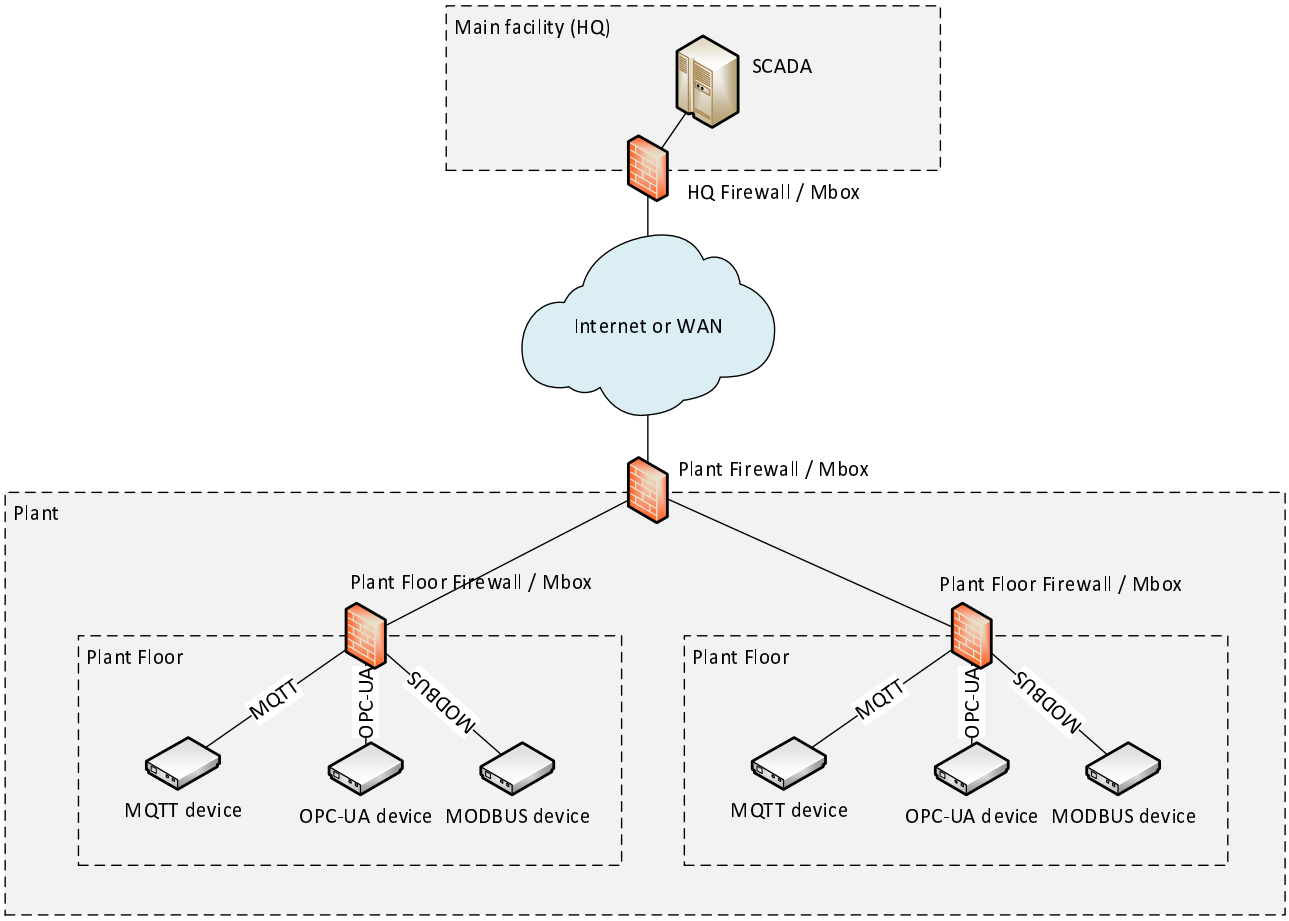


Figure 16: Conceptual illustration of middleboxes at different levels in an industrial control system structure

The protocols involved in this use case usually reside on top of TCP, and in some cases (e.g. MODBUS) on UDP in the protocol stack. Therefore, the principles in clause 8 for adding TLMSP support in PLCs and SCADA software.

6.8.4 TLMSP Gateways

Leveraging TLMSP point-to-point would require all devices to implement support for TLMSP. This includes every control device used on the plant floor having limited computing power, and most importantly, these devices can be difficult to update for a foreseeable future. Also, in the context of industrial control systems, encrypted communication on the plant floor is very often not desirable since it introduces delays, key distribution problems and other aspects that could lead to very unpleasant and costly system downtime.

Therefore, it is suggested that TLMSP is introduced at the exit point of the plant floor, i.e. in the firewall separating the plant floor and plant zones in Figure 16. This firewall would therefore, apart from monitoring traffic that enters and exits the plant floor, also act as a TLMSP Gateway and tunnel traffic over TLMSP. TLMSP could carry traffic from the plant floor firewall through the plant firewall, over the internet to the HQ firewall and SCADA system. Monitoring and control could be performed at every zone border according to ISA/IEC 62443 without doing any modifications on any of the plant floor devices.

6.8.5 Discussion

As shown in this clause the concept of middleboxes and TLMSP allows for the otherwise conflicting combination of encrypted communications over security zones with requirements on *monitor and control* from ISA/IEC 62443 series of standards. A drawback would be the fact that support for TLMSP is required in all nodes communicating over zone borders but this could be somewhat mitigated using TLMSP Gateways between a group of devices and destinations servers. The devices themselves would in this scenario communicate using unencrypted communications though the TLMSP Gateway towards the destination server.

6.9 Lawful Intercept

6.9.1 Use case description

6.9.1.1 Introduction to Lawful Intercept

Lawful Interception (LI) refers to facilities in telecommunication networks whereby Law Enforcement Agencies (LEAs), after legal authorization, can obtain a copy of communication content or metadata related thereto, of relevance to investigate or prevent serious crime. Intercept is performed at a few, well-defined Points-Of-Intercept (POIs) integrated in the network architecture via 3GPP standards. The intercepted traffic or metadata is then made available from the Communication Service Provider (CSP) to the LEA over protected *handover interfaces*, which are standardized in ETSI. Most countries in the world have legal frameworks requiring CSPs to assist LEAs in this manner. The authorization is commonly in the form of a court order and usually identifies an explicit target for the intercept (an identified individual). Exactly what is to be understood as a CSP varies across countries, but in most cases, at least the traditional telcos and mobile network operators fall under LI obligations.

The LI environment has increased in complexity over the years as an effect of the rapidly evolving eco-system and technology, in particular the mobile network eco-system:

- Mobile subscribers (including those with criminal intent) are today highly mobile and can obtain services from mobile operators while visiting other countries, based on roaming agreements between a home CSP (hCSP, handling the subscription) and the visited CSP (vCSP, where the subscriber is currently located).
- Communication services, traditionally provided by the mobile operator itself (e.g. voice), is receiving strong competition from 3rd party service providers (so called over-the-top, OTT, services) out on the internet using smartphone apps.
- The OTT services tend to apply encryption to a high degree and solutions are being standardized by which also MNOs more easily can offer encrypted services, e.g. [i.18].
- Depending on the legal framework, the OTTs might not fall under the same LI obligations as the MNO transporting the application traffic.

In a non-roaming scenario and where the service is provided entirely by the hCSP, things are reasonably straightforward: there is only one jurisdiction involved (where the hCSP is located) and it does not matter if the service is encrypted or not since plaintext content is usually available at the hCS. In most countries, the CSP is required to remove any encryption which it has been responsible for providing in the first place.

In the roaming scenario, two different MNOs are involved in delivering services and these could be located in different countries/jurisdictions. A mobile subscriber can therefore in principle be of interest for LI in one or both of the jurisdictions involved. Here, a jurisdiction cannot be made dependent on the need to explicitly request support from another jurisdiction in order to perform LI. This would have the undesirable effect of revealing the identity of LI targets across jurisdictions. A special challenge is when the service is encrypted and anchored at the hCSP: this will make it difficult for the jurisdiction in which the vCSP is located to effectively utilize LI, see left half of Figure 17 below. The LEA in the jurisdiction of the vCSP could present the vCSP with a warrant, but the LEA would only be able to obtain encrypted IP packets. This is today commonly handled by signing roaming agreements where the hCSP agrees to not enable the encryption during roaming. This satisfies legal requirements but is unfortunately a sub-optimal solution from the end-user point of view.

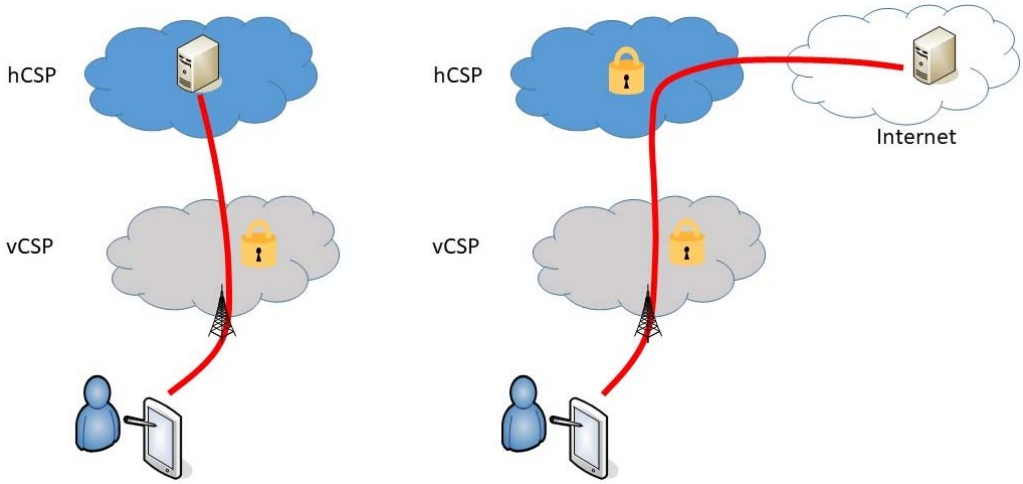


Figure 17: Roaming-scenarios which are problematic from LI point of view in vCSP (and sometimes also in hCSP)

An even more complex case is the aforementioned roaming scenario where, additionally, the communication service is provided by an OTT (potentially in a 3rd jurisdiction), and potentially without the same stringent LI obligations. This could make LI problematic in both hCSP and vCSP. This is also shown in Figure 17, right half.

As mentioned, there is currently ongoing standardization work referred to as Authentication and Key Management for Applications (AKMAs), [i.18], that would in principle enable users to utilize services with strong encryption in both of the aforementioned roaming cases, based on using the USIM-card as a generator for encryption-keys. This is applicable in both of the roaming scenarios of Figure 17: in the OTT case, the OTT could obtain the keys from the hCSP. However, except for the non-roaming case, a technical LI-solution is not yet defined, making it necessary to still fall back to regulating the use of encryption via roaming agreements.

In this clause, a high level, middlebox-based architecture is proposed which could serve as a basis for an LI solution while allowing encryption. The middleboxes serve as POIs at the vCSP and hCSP and are assumed given access to decryption keys by the hCSP.

6.9.1.2 AKMA Overview

AKMA is leveraging the key generation functions of the USIM: Normally, this highly secure key generation environment is only used to derive keys to protect the access (radio) interface, but with AKMA, keys usable by various applications can also be derived. An AKMA Anchor Key (K_{AKMA}) is derived by the USIM and the hCSP, and a copy of this key is stored at an AKMA Anchor Function (AAnF).

NOTE: USIM uses a long-term key, pre-shared with the hCSP. From this key, a network level base key is derived as result of basic USIM-network authentication. This base key, in turn, is used to derive the AKMA-keys locally at the USIM and hCSP without need for additional information exchange.

Various applications (in hCSP, vCSP, or even at an OTT) can request application-specific keys (K_{AF}) which are then derived from the K_{AKMA} and provided to the respective applications. This is done in such a way that a key-compromise at one application would not impact other applications, even if those keys depend on one and the same K_{AKMA}. This is shown in Figure 18. The application specific keys can then be used together with a suitable security protocol to establish encrypted sessions between the user device and the application. In the present document it is shown how to leverage AKMA in the case when this protocol happens to be TLMSP. As will be seen, the issues related to key-transfer across CSPs is largely alleviated when using TLMSP.

AKMA provides identifiers for the AKMA keys (denoted A-KID) and for the AFs requesting the key(s).

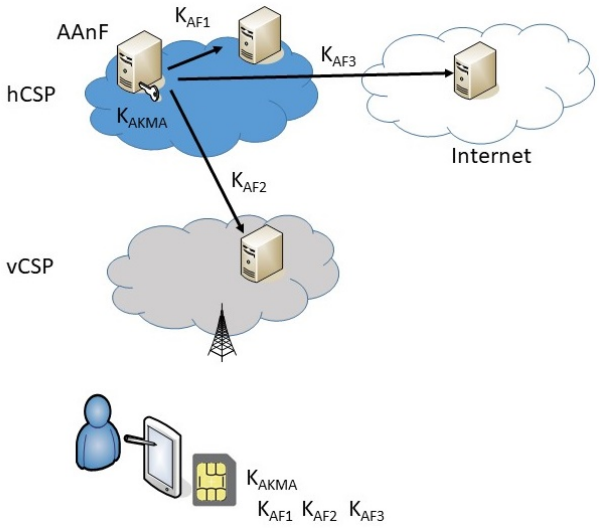


Figure 18: AKMA architecture

6.9.1.3 Security objectives, Threats and Trust Model

Obviously, there are threats related to the authenticity of the LI warrants, the security of the handover interfaces etc, though these are not really middlebox-related and are assumed handled securely by the LI-part of the network. The main middlebox-specific threats are related to potential unwanted exposure of keys and/or user traffic which is manifested as assurance requirements on the middlebox implementation, which makes this a main security objective, though falling outside the scope of the present document.

From trust model point of view, since it is assumed that the technical solution of ETSI TS 133 535 [i.18] is used to provide encryption keys, these keys are therefore the responsibility of the hCSP which needs to be assumed to handle them appropriately with regards to their exposure to unauthorized parties. Similarly, the USIM needs to be assumed residing in a secure, tamper resistant element. Since LI in the vCSP is normally dependent on obtaining (decryption) keys from the hCSP, it is assumed that the hCSP does not on purpose attempt block the vCSP from its LI obligations, i.e. by providing incorrect keys (or no keys at all). Due to requirement that activation of LI at one CSP should not be detectable by another CSP, it usually needs to be assumed that the hCSP provides keys for *all* of its customers roaming into the vCSP network, regardless of whether or not they are targets for LI at the hCSP. With TLMSP, most of these issues vanish.

When an OTT is involved in providing services, it further needs to be assumed that the OTT does not attempt to hinder the CSPs from their legal obligations. This could happen if the OTT further modifies the AKMA K_{AF} key, attempts to use another key and/or protocol, etc. There exist some means to handle this, since the AAnF needs to be able to authenticate and authorize the OTTs that are permitted to obtain AKMA keys.

6.9.2 Technical characteristics

The service in question is assumed to be a communication service provided by an application function AF. The service could be in the form of messaging, chat, or IP multimedia (voice + video) provided by an OTT or the hCSP to a subscriber roaming in a vCSP network. Since the former case is the most complex, it is the focus of the discussion, see Figure 19. Thus, mobile clients (e.g. smartphones) equipped with USIMs on secure elements (potentially instantiated via eSIM) are assumed in use. The hCSP deploys an AKMA AAnF server generating keys for the service.

Regarding communication protocols, the communication service AF follows the AKMA protocols to obtain (encryption) keys from the AAnF. The application layer protocol for the communication flow could be HTTTP or something else, depending on the type of media carried. In any case, it assumed that TCP transport is used and that TLMSP is present in the clients, the middleboxes, and the AF, placed between the application and the transport layer. It is assumed that the AKMA key K_{AF} is used as a pre-shared key between the client device and the AF, serving as basis for TLMSP authentication and session key-establishment.

Since there could be need to perform LI independently at hCSP and vCSP, two middleboxes, hM and vM are present in these two networks. The middleboxes provide LI POI functionality and assuming a 5G network, hM and vM could be collocated with the User Plane Functions (UPFs) present in these two networks.

Normally, with most other middlebox approaches, there would be a need to distribute AKMA keys also to the middleboxes (the dotted arrows in Figure 19). A big advantage of TLMSP is that this is no longer needed, because the middleboxes take part of the client-AF TLMSP handshake and will be given access to TLMSP session keys in-band. Thus, the pairwise TLMSP keys (shared between an endpoint and each of the middleboxes) can be derived by Diffie-Hellman or some other mechanism, they do need to depend on K_{AF} or any other AKMA keys. Thus, K_{AF} would typically only be used for client-server authentication purposes. The same holds for other necessary information, such as which encryption algorithms to use, etc.

LI warrants could be service-specific, and for the purpose of the use-case, warrants are assumed to cover the service provided by the AF in its entirety.

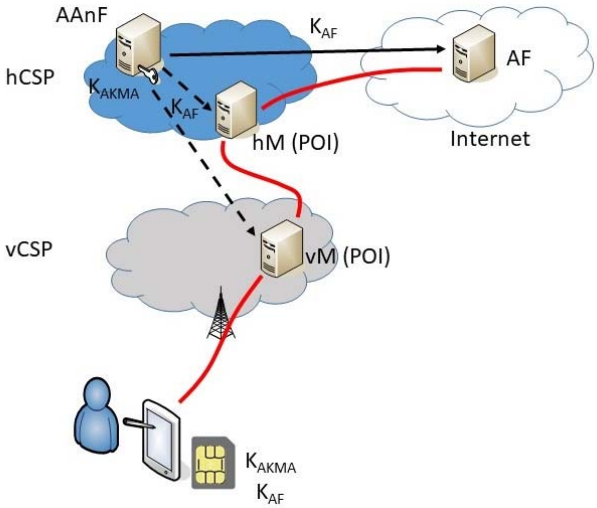


Figure 19: AKMA roaming LI-architecture, in the most general case

6.9.3 TLMSP mapping

The clients are assumed configured to initiate TLMSP to protect the application traffic by indicating, in the TLMSP middlebox list extension, identifiers for vM and hM. These could follow some generic naming scheme, e.g.:

mbox_list = { TLMSPmbox.vCSP. 3gppnetwork, TLMSPmbox.hCSP.3gppnetwork.org }

The warrants cover the full service provided by the AF. Thus, the middleboxes are also in the handshake signalling assigned read-privilege for the entire service data flow. This needs to be done regardless of whether LI is activated or not (since the client will not, and ought not, know about LI). The middleboxes do not require, and ought not to be assigned privileges beyond that. This is sufficient since it enables any of the two POIs to (independently) decrypt traffic and (via handover interfaces) making it available to LEA "en claire". Again, since the warrants are assumed to cover a specific service in its entirety, there is only need for one TLMSP context, covering the whole application payloads and headers.

It is of interest to note that if there is no warrant (or the TLMSP-protected service is outside the scope of the warrant) the middleboxes will still remain on path and forward all traffic.

There could of course be other services which also uses TLMSP, e.g. services using connections egressing from the mobile infrastructure to an enterprise network. Such flows will however not include the middleboxes vM and hM in the set-up signalling and, vM/hM would not even participate in such sessions.

6.9.4 Discussion

While as mentioned, the middleboxes will have the possibility to intercept all TLMSP-flows passing them, so called *over-collection* needs to be avoided. Over-collection occurs, not due to the decryption capabilities as such, but rather if/when the LEA is handed over information which is not in scope of a warrant. Thus, this boils down to determining which sessions that are to be handed over to LEA. This requires a mapping from the scope of the warrant to the type and identity of the service provided by the AF and is a standard LI-issue addressed by LI-specifications and therefore outside the scope of the present document.

Observe that the middleboxes need to remain on path even when LI is not being activated. If not, the client might, e.g. from network delay measurements be able to deduce whether interception is taking place or not.

Figure 19 focuses on the most complicated situation: roaming in combination with AF at an OTT. A few words could be said about the simpler cases:

- For roaming with AF at hCSP, only the vCSP middlebox vM is needed. While one could envision also a hM middlebox at the hCSP, it appears much simpler to implement LI at the hCSP directly in an AF-specific POI collocated with the AF.
- Similarly, for roaming with AF at vCSP, no middlebox is needed. TLMSP could still be used, or, the automatic TLS_fallback mechanism defined in the TLMSP specification could be used. This is a form of local break-out which will not allow the hCSP to see the traffic. Whether such a setting is compliant with LI requirements on the hCSP depends on national law. If it is not permitted, it is assumed that roaming agreements are used to handle this.

Finally, it needs to be acknowledged that many communication services are likely using UDP transport. Thus, a UDP-version of TLMSP would need to be defined to be relevant for LI relating to such services.

7 Protocol Specifics

7.1 General

7.1.1 Introduction

An observation is that many protocols in the present document share some common concepts such that of ordered arrays, (also referred to as lists), and objects, (also referred to as associated array, or dictionary, etc.). The purpose of this clause is to have a general discussion around these concepts. Some of the protocols might have attributes on objects, and some can have descriptions such as schemas.

An array might look like:

```
array               ::= begin-array [ element *( element-separator element )]
                        end-array
    element         ::= object | array | simpleElement
    simpleElement   ::= string | number | …
```

Some explanation might be in order. The description above is in "BNF-style" format. Thus, the square brackets [] indicate that the enclosed elements are optional. The *() symbol means zero or more occurrences within the parenthesis. A | B, means A or B.

A typical element-separator used in a protocol would be a comma sign. In some protocols, which are line based, the element-separator would be a newline, or a newline plus some extra character.

Typically, the ordering of the (ordered) array matters. So for instance:

```
begin-array element1 element-separator element2 end-array
```

is not the same as:

```
       begin-array element2 element-separator element1 end-array
```

White space might, or might not, be important.

An overall description of an object might look like:

```
object  ::= begin-object [ member *( member-separator member )  ] end-object
member  ::= string name-separator value
value   ::= object | array | simpleElement
```

Typically, an object does not have a pre-determined ordering of its members. For instance:

```
begin-object member1 member-separator member2 end-object
```

would in general represent the same object as:

```
begin-object member2 member-separator member1 end-object
```

A way of describing the handling arrays and objects in a TLMSP or other context based protocol would be to indicated the mapping to a certain context c_i by writing $c_i(\ldots)$, or to use a "colouring" of elements and members (an example of this is shown in clause 7.4.3). Perhaps, at the application layer, those would not be visible, rather be present in some intermediate TLMSP adaptation layer as discussed in clause 9. This adaptation layer would then take care of placing the $c_i(\ldots)$ part in a container associated with context c_i. At the same time, one would not want the application layer (in a middlebox) to be presented with something which a parser would consider to be an invalid protocol message. For instance, a middlebox which has read access to an element within an array is also supposed to have read access to the begin-array and end-array tokens. This approach would allow reuse of an existing parser, unaware of TLMSP.

In TLMSP, contexts is a major concept. However, also containers, which are the actual bearers of data associated with contexts, are important. Each container is equipped with a context identifier. Another major concept is that of permissions, e.g. read/write/delete. Say that a middlebox has read access to some part of a protocol message, an array for instance, but that it finds it necessary to include a new element within that array. If a middlebox M has the right to include data within context B, and the array is in a container associated with context A where M has only read permission. Then M would only be able to include data into the array provided there was a container boundary within the array. Therefore, it is not sufficient to just be able to specify a context identifier. It would also be necessary to be able to specify container boundaries and align to those.

An array in a context-aware setting would look like:

```
contextArray ::= c(begin-array) [c()]  [ c(element) *(c( element-separator
                     element ) | c() ) ] c(end-array)
```

That is, the begin-array and end-array belong to some context, there might be an empty container, or a container-boundary, c(), directly after the begin-array, and those could also be present anywhere between the elements. Typically, if two or more elements are adjacent, they would end up in the same container. There is no need for unnecessary container boundaries, if not asked for.

NOTE: An element could be an array or an object, and in that element there could be other contexts present. Typically, the begin, and end markers of the element would then belong to the context specified in c(element).

EXAMPLE: Assume there are three contexts c_1, c_2, c_3. In this example the container boundary is coded as $c_0()$, because context c_0 is reserved (e.g. accessible by all). The begin-array, end-array, and elementA is in context c_1, the elementB in context c_2. Equivalently, $c_i(\ldots)$ can be understood as placing the associated data in a container associated with context c_i. and there is an empty container with context c_3 at the end of the array.

```
c1(begin-array) c0() c1(elementA) c2(element-separator elementB) c3() c1(end-array)
```

In the example above, a middlebox with only read access to c_1 can parse and read elementA, but nothing more. A middlebox which also has write access to c_2 can modify elementB and also add further elements after elementB as long as they are placed in context c_2.

Similarly, for objects:

```
contextObject ::= c(begin-object) [c()]  [ c(member) *(c(member-separator
                      member) | c()) ] c(end-object)
```

Some remarks are in place. For any structure of the form:

```
begin-token [ item *( item-separator item )] end-token
```

formatted into contexts as:

```
c(begin-token) [c()] [ c(item) *( c(item-separator item ) | c()) ] end-token
```

the following can be said:

- The suggestion is that any middlebox which has at least read access to some item also has at least read access to the begin- and end-token:
  - Even further, the suggestion is that the begin-token and end-token belongs to the same context.
- A context unaware application layer is presented with grammatically correct placement of the item-separators. That means that:
  - If a middlebox inserts one or more items at the beginning of a token, it is preferred to finish its operation by inserting an item-separator, unless the token is empty.
  - If a middlebox does not have read access to the first item, it ought to remove the misplaced item-separator before parsing the token, or include an empty item if that makes sense to the application.
  - The container boundary between an item and an item-separator ought to be placed so that the container data begins with the item-separator, leaving any optional white space in the previous container.
- If a middlebox has read access to some context in the middle of the object but not on the level above, the effect could be that the hierarchical structure of the object gets corrupted. This might, or might not be a concern. The suggestion anyway, is that the adaptation layer ought not to allow mappings where some middlebox has read access right down in the hierarchy of the object without also having access to all of the higher object layers.
- For objects, the ordering of its members need not be well-defined at the application layer. The serialization of the object ought to group members according to the contexts they belong to. If possible, making sure that all middleboxes has read permission to the context of the first member, if any.

7.1.2 Additional remarks

An equivalent definition of arrays and object would be:

```
array   ::= begin-array [ *( element element-separator ) element ) ] end-array
object ::= begin-object [ *( member member-separator ) member ) ] end-object
```

with formatting into contexts as

```
contextArray2 ::= c(begin-array) [ *( c(element element-separator) | c() )
                     c(element) ) ] [c()] c(end-array)
contextObject2 ::= c(begin-object) [ *( c(member member-separator) | c() )
                     c(member) ) ] [c()] c(end-object)
```

The difference from clause 7.1.1 is that it is the last element which when invisible to a middlebox could cause a misplaced item-separator at the end of the array or object.

Both approaches to context aware tokens ought to be useful, even within the same protocol, and depending on the circumstances be put to use.

7.1.3 Attributes

Some protocols allow for attributes on their objects. The recommendation is to place all attributes in the same context as the begin-object and end-object. If there is a context attribute, then place all the members in this context, however not the begin-object and end-object. Naturally, as mentioned before, a member could have a context aware structure.

7.1.4 Schemas

Some protocols allow for schemas. When a schema is used, it ought to be possible to add a context to it in the same manner as for instance the type of an item. When an object later appears, the corresponding item could be place in a container with that specific context. For an example of contexts within schemas, see example 1 in clause 7.4.3.

7.1.5 Line based protocols

Some protocols are structured in lines. It ought to be beneficial to place container boundaries at line breaks for those. That is, to begin the data of a container at the start of a line. Naturally, within a line there could also exist reasons to change the context or to introduce a container boundary.

7.2 Overall structure

In the sequel, a number of specific protocols are discussed. Each protocol clause follows a common template describing:

- An introduction to the protocol.
- A feasibility discussion if it is possible to use the protocol with TLMSP:
  - Can the general approach be taken?
  - Do arrays and objects exist in the protocol?
  - Does the protocol use attributes on objects?
  - Are there schemas in the protocol?
  - Is the protocol line based?
- Using the protocol with TLMSP.
- Examples, if not part of the using part.

7.3 HTTP

7.3.1 Introduction

HTTP, forming the basis for the world-wide web is probably the most used application protocol. HTTP messages have a header and possibly a body. The header mainly consists of a start line and some header fields. The body is not always present and sometimes not even allowed. In IETF RFC 2616 [i.8], the HTTP message structure is described as follows:

```
HTTP-message   ::=  start-line
                   *( header-field CRLF )
                   CRLF
                   [ message-body ]
```

Looking at the HTTP header, it starts with

```
start-line     ::= request-line | status-line
  request-line   ::= method SP request-target SP HTTP-version CRLF
status-line    ::= HTTP-version SP status-code SP reason-phrase CRLF
header-field   ::= field-name ":" OWS field-value OWS
```

In the past days, HTTP field-values could for readability include what is now called "bad" whitespace (specifically, `CRLF ( SP | HTAB ))`. Bad whitespace is today deprecated and header-fields are supposed to terminate when a `CRLF` is encountered. That means that senders will not insert bad whitespace, but receivers still need to parse for it.

7.3.2 Feasibility of using HTTP with TLMSP

The HTTP header matches the general approach by identifying:

```
begin-array [ *(item  item-separator) item )  ] end-array
```

with:

```
start-line [ *( header-field CRLF ) header-field) ] CRLF CRLF
```

However, it seems also feasible to use the line-based structure of the HTTP header and use the line based approach from the general clause.

There are also parts in the HTTP header with an object structure. Techniques from the general clause ought to be possible to use for them.

The HTTP body is more of a "blob" in the protocol. However, the body is often populated by popular protocols where usage of TLMSP features would be possible as described later in the present document.

7.3.3 Using HTTP with TLMSP

7.3.3.1 Header fields

A likely use of TLMSP is to control read and write/modify access to HTTP headers and HTTP body and/or different fields within the header. Header fields are easily separated from each other and the parsing rules seem to be very helpful when dividing a header into TLMSP containers as long as the header fields are mapped into them. Note though that for middleboxes who are assumed to also access the message body, there is a need to ensure that such a middlebox also has access to those parts of the header that are needed for parsing the body.

If some middlebox is assumed or required to include a header field at a certain position among header fields (for example, a control header-field that ought not to be placed at the end, see below), the boundaries of the containers needs to be placed to allow insertion of another container there, using a context with assigned write-privilege, or, the corresponding part of the header should be in a context writable by that middlebox.

When it comes to contexts within a header field (i.e. a context only covering part of a header field) more care is needed.

In essence one could think of an adaptation layer which searches for specific field-names and places the corresponding header-fields or part of the field-value into contexts, or empty context holders just before or after certain header-fields. Although IETF RFC 2616 [i.8] demands the receiver to expect arbitrary ordering of the header-fields which necessitates the receiver to read all header-fields before taking any action, there are custom practices on the sender. For instance, so called control header-fields ought to come first. In the old HTTP 1.1 IETF RFC 2616 [i.8], the header was divided into a general header, a request/response header and an entity header.

7.3.3.2 Location Headers

As a result of e.g. a HTTP PUT-request, a resource of some kind might be created. In the response, the status-code 201 is then used together with a location header with the field-name "Location". The server could also include the actual object. Assume that the object is more suitable to be stored in a middlebox, or cached (and there is no contradicting cache-control from the server, see below). Then the middlebox would typically want to change the location header. In such scenarios the location header is preferably writable by the middlebox and the object at least readable.

Other situations in which a location header may be present is in redirections, with status-code 3xx. Also here a middlebox could have the need to change the location in the response.

7.3.3.3 Authentication

7.3.3.3.1 HTTP native

There are header-fields of interest for authentication purposes.

The request field-names for challenges are:

```
Authorization and Proxy-Authorization
```

and response field-names for credentials are:

```
WWW-Authenticate and Proxy-Authenticate.
```

As an example of the usage within the context of TLMSP, assume that the middleboxes know that the server will perform an authentication and moreover, the middleboxes accept to send along the request unauthorized.

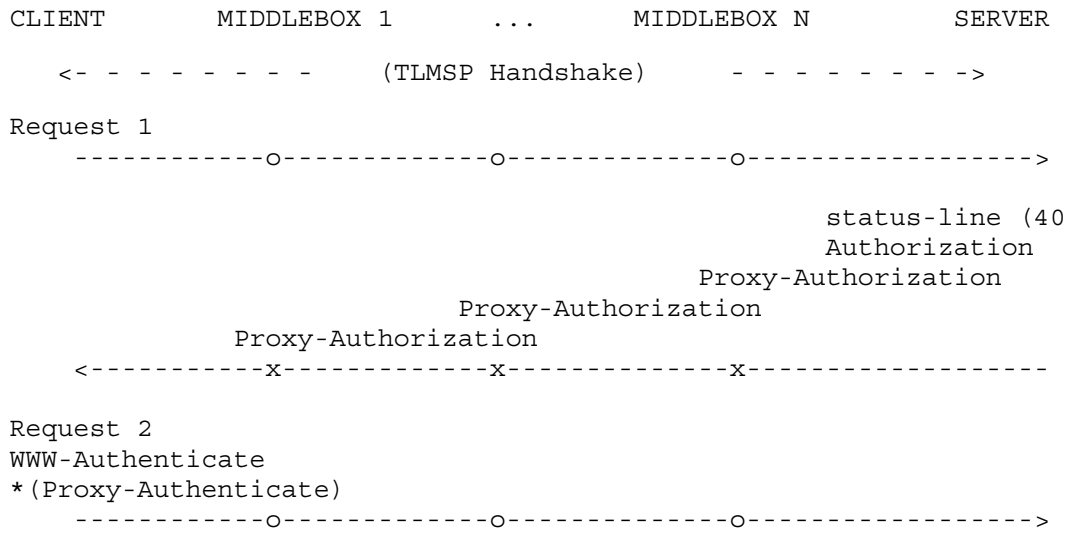


Figure 20

Actually, only the so called `Set-Cookie` header-field is allowed to occur a multiple of times in the HTTP header [i.8]. Perhaps in real life, only one middlebox would perform Proxy-authentication so it might not be an issue. In any case, according [i.9] there is a realm included so separating the different proxy authentications would not be impossible. Following practice, the authentication headers belong to the request/response header. The server needs to at least make sure there is a suitable container boundary, or include a container with no data, associated with a writable context, for the middleboxes to include their `Proxy-Authorization` header fields for instance following the general header. The wisest approach could, despite all, be to break the custom and use the possibility given by [i.8] to freely position header-fields. In that case the server could create the HTTP header, then insert one or more empty containers for middlebox use just before the ending `CRLF` of the header.

There are problems with the last example. The middleboxes cannot in general know that the server will request authorization from the client. Also, the request may change the state on the server in a way the middlebox was there to prevent, e.g. by uploading a blog post or whatever.

One solution could be for middleboxes to only allow unauthorized harmless things like a minimal `OPTIONS` request and hope for the server to respond with a writable header. There is the possibility to include a path in the `OPTIONS` request so that the server can tell if user authorization is needed or not. The middlebox might want to check that not too much information is sent.

EXAMPLE:
```
OPTIONS * HTTP/1.1
OPTIONS /public HTTP/1.1
OPTIONS /user HTTP/1.1
```

The `OPTIONS` method is defined as safe and idempotent, meaning that the server state does not change and the response needs to always be the same. That is somewhat untrue if the server generates a unique challenge which it also should store to avoid replay attacks. However, the challenge belongs to the session and can be dropped when the session is closed so in that respect the method is still safe. Since it is allowed to include a date in an idempotent response, a challenge is also allowed.

The server typically either respond with a `204 No Content`, an `Allow` header-field, etc., if it does not require authentication, or a `401 Unauthorized` with an `Authorization` header-field, otherwise.
The adaptation layer preferably then give the possibility for the middleboxes to insert their `Proxy-Authorization` header-fields.

Some recommendation for HTTP Digest:

- A middlebox set-up for HTTP digest user authentication ought to respond (if it has write permission) with `407 Proxy Authentication Required`, (or in a hopeless case `401 Unauthorized`), if the client's request is something needing authorization and the request did not include valid credentials. It then either delete all content from the client in the server direction or if it is unable to, terminate the session.

- If the client receives a `401 Unauthorized` containing one or more `Authenticate` or `Proxy-Authenticate` and it has the relevant credentials, it ought to send another request containing those credentials for all the requested authentications. If there are no `Authenticate` or `Proxy-Authenticate` header-fields, the client gives up.
- If the client receives a `407 Proxy Authentication Required`, it, if necessary, opens a new session (if it was terminated), and sends an `OPTIONS` request in order to get/request new challenges.
- In HTTP Digest there is the possibility of nonce reuse, using a nonce counter. Depending on configuration, there is a limit on how many reuses are allowed. The limit ought to be as high as reasonably possible. The client requests new challenges for all entities at the same time.

7.3.3.3.2 OAUTH

OAuth 2.0 [i.13] uses HTTP transport and a lot of redirections are used. Nevertheless, using OAuth on the server does not seem to be a problem, possibly also combining with HTTP Digest authentication on the proxies.

Using multiple OAuth authentications, involving client to middleboxes as well, seems doable but complicated. In the first response a number of location headers to Authorization Servers could be present which perhaps is not such a big issue. However, the redirections do not stop there. The chain of redirections continues back all the way.

7.3.3.4 HTTP Request

The method of the request-line may be one of `GET/POST/PUT/PATCH/DELETE/OPTIONS` (this is not an extensive list). The request-target can have a number of formats. The one of interest here has the format:

```
https-URI = "https:" "//" authority path-abempty [ "?" query ] [ "#" fragment ]
```

One could think of having the path, or parts of it, in a specific context not the same as for the authority part. Anyway, the path is easily parsed. If there is a question mark present in the http(s)-URI, it needs to be followed by the query. The query is terminated by whitespace or the '#'. The query will typically consist of fields which could be interesting to put into different contexts. There are a number of formats for the query, some of which will be described later.

Regarding queries, the query usually has the structure `member *( value-separator member )` of clause 7.1 with:

```
member                ::= string name-separator value
value-separator       ::= '&'
name-separator        ::= '='
```

There are no begin-objects, begin-arrays, etc. That is because the start of the query is not contextually a part of the query. However, in a sense '?' could be seen as the begin-object and '#' or SP as the end-object. The value-separator for arrays is ','. An approach to context mapping of the query is to use the same as the approach on general objects.

In the query, URL encoding is used, and many characters are forbidden. Values could be JSON objects. In the mapping of these it is noted that due to the URL encoding:

```
begin-object      ::= %7B
end-object        ::= %7D
value-separator   ::= %2C
```

7.3.3.5 The HTTP body

The message body is by IETF RFC 2616 [i.8] supposed to start directly after an empty `CRLF` line. This marks the end of the header as well. A simple context/container mapping could be to place the header in one context/container and the body in another. The rule ought then to be to include the terminating `CRLF` in the context/container where the header is. Otherwise, a header parser might misbehave. However, already with this simple approach problems could arise while the header is very structured, indeed only US-ASCII is allowed, the body is considered to be a blob. The size of the body can be deduced either explicitly by a `Content-Length` header-field or indirectly by header-fields telling which type of encoding is used. In a body where contexts have been used so that middleboxes either cannot read all of the body content, or can write in the body but not able to change the `Content-Length`, a `Content-Length` header field is preferably avoided.

In applications using several contexts for the body, probably the best approach is to use a multipart `Context-Type` header field, for instance `multipart/mixed` or `multipart/json`. In multipart encodings there usually is a unique encapsulating boundary. There could be a preamble that needs to be ignored and an epilogue equally to be ignored. Usually, multipart can be nested as long as the boundaries differ.

Even for multipart bodies, the `Content-Length` header-field can be used in HTTP. As noted, it needs to be suppressed in a context mapping application.

7.4 JSON

7.4.1 Introduction

JSON is a very popular protocol which is human readable and easy to machine-parse and is also considered light-weight. JSON is built up by name-value pairs and ordered lists (or arrays).

7.4.2 Feasibility of using JSON with TLMSP

If one looks at the definition of a JSON object, it looks like:

```
object ::= begin-object [ member *( value-separator member )  ] end-object
member ::= string name-separator value
```

recognized from the general section when identifying:

```
begin-object      ::= '{'
end-object        ::= '}'
value-separator   ::= ','
name-separator    ::= ':'
```

Likewise, an array has a similar structure:

```
 array ::= begin-array [ value *( value-separator value ) ] end-array
```

when identifying:

```
begin-array       ::= '['
item-separator    ::= ','
end-array         ::= ']'
```

There are no attributes in JSON. However, schemas do exist.

7.4.3 Examples

There are two contexts, c_1 and c_2, and the wish to use JSON in a multi-part HTTP body. A schema for the JSON object is put in the preamble followed by the starting curly brace '{' of the object, continue with the object parts, finally ending with '}' in the epilogue. In the example the contexts' content are enclosed in $c_1()$ and $c_2()$.

EXAMPLE 1: {"user":"Alice","realm":"…","password":"secret"} with "user" and "realm" in context c_1 (which all can read) and "password" in context c_2 is mapped into

```
c1(
Content-Type: multipart/json; boundary: ABCDE
{
   "$schema": "…",
   "title": "Example",
   "type": "object",
   "required": ["user","realm"],
   "properties": {
      "user": {
         "type": "string",
         "x-context":1
      },
      "realm": {
         "type": "string"
         "x-context":1
      },
      "password": {
         "type": "string",
        "x-context":2
      }
   }
}
{
--ABCDE
   {"user":"Alice"},
   {"realm":"www.example.com"}
)
c2(--ABCDE
,
   {"password":"secret"}
)
c1(--ABCDE—
}
)
```

EXAMPLE 2: This example is the same as example 1, but where one would additionally like the "realm" attribute to be modifiable. To this end, a third context c_3 could be introduced (with write privilege for authorized middleboxes) and the part:

```
c1(
Content-Type:
…
{--ABCDE
   {"user":"Alice"},
   {"realm":"www.example.com"}
)
```

could be replaced by:

```
c1(
Content-Type:
…
{--ABCDE
   {"user":"Alice"},
   {"realm": ) c3(www.example.com) c1( }
)
```

Here, it does not matter if some middleboxes do not have read access to c_3 since the message is still possible to parse correctly. (Since "realm" is a required attribute, it ought to be readable as part of c_1, even if the actual value of realm is not.)

EXAMPLE 3: In this example, the colouring scheme of clause 7.1.1 is used. There are four contexts 1, 2, 3, 4. Thus, data showed in for example red color is to be understood as being placed in context 2, and so on. Keywords "a","w" are mapped to 1, "b" to 2, "c","v" to 3 and "d" to 4. Moreover, values in "d" are mapped to context 4 if they are one-digit numbers and context 3 otherwise. (A colored _ means space in this example).

{"a":1,"b":2 ,"c":{"v":"V","w":"W"}, "d":[1,2,3,4,11,5,6,12,13,2]} →
{"a":1,"b":2_,"c":{"v":"V","w":"W"},_"d":[1,2,3,4,11,5,6,12,13,2]} →
{"a":1,"b":2_,"c":{"v":"V","w":"W"},_"d":[1,2,3,4,11,5,6,12,13,2]}

Someone with read access to contexts 1, 3, 4, but not 2 would then see:
{"a":1,"c":{"v":"V","w":"W"}, "d":[1,2,3,4,11,5,6,12,13,2]}
which makes perfect sense. However, someone with read access to all but context 3 would see
{"a":1,"b":2_,"w":"W", "d":[,11,12,13]} →
{"a":1,"b":2_,"w":"W", "d":[11,12,13]}
which is a properly formatted object but could potentially be devastating. If all but context 4 can be accessed it would result in a totally corrupt object
{"a":1,"b":2_,"c":{"v":"V","w":"W"}1,2,3,4,5,6,2}

7.5 YAML

7.5.1 Introduction

YAML includes JSON, so its constructs are allowed to be used and methods from the previous clause could be used. Apart from that, it is a line based protocol and structure is achieved using white space indentation. Arrays are called lists and objects associative arrays. Lists can be written using JSON, or by putting each element on a separate line with indentation and hyphens.

EXAMPLE: An array with three objects, the value of TCP is an object with two members, the value of TLMSP is an array of three `simpleElements`, and the value of HTTP an array of two `simpleElements`

```
- TCP:
    port: 80
    IPv4: 192.168.2.1
- TLMSP:
    - containers
    - contexts
    - security
- HTTP: [header, body]
```

7.5.2 Feasibility of using YAML with TLMSP

When using JSON within YAML, the approach from the JSON section is certainly valid. Otherwise, making sure to place container boundaries at line breaks ought to do the trick. There exists schemas in YAML, but no attributes on objects.

7.5.3 Using YAML with TLMSP

It seems straightforward to use TLMSP together with YAML. Since tokens have hierarchy dependent begin tags, end tags, and separators it ought to be easier to adapt to the line based approach.

7.6 A glance at OpenAPI

The Open API Initiative (OAI) is an industry initiative to provide an open source community for and a standardized way of building API:s. The OpenAPI specification is used among others in the 3GPP 5G specification.

An API in [i.14] is described in JSON objects, specified in JSON or YAML. This is best viewed in [i.14]. However, parameters are among others described by their *type* and perhaps *format*. For instance, the type may be integer and the format `int64` to further specify which kind of parameter it is.

EXAMPLE: Description of a required username query parameter of string type. Added to this is a suggested "x-context" property.

```
{
    "name": "username",
    "in": "query",
    "description": "username for the user accessing the service",
    "required": true,
    "schema": {
        "type": "string",
        "x-context": 1
    }
}
```

Moreover, parameters can be referred to in other parts of the object using a {parameter} syntax. For instance, if in the above example there had been "in":"path", the username could appear in a path description "…/{username}/property". In a similar fashion, it would be prudent to be able to specify "context": {parameter}. Other choices for "in" are header, and cookie:

- OpenAPI seems to be a good candidate to be approached from the TLMSP group:
  - Structured and standardized.
  - Widely adopted.
  - The API is for all ends, or infrastructure.

7.7 XML

7.7.1 Introduction

In XML, there is the concept of a tag. A tag could be a start-tag, an end-tag, or an empty-element tag. An element is a component which starts with a start-tag and ends with an end-tag, with its content in between:

```
start-tag ::= <string>
end-tag   ::= </string>
empty-tag ::= <string/>
```

EXAMPLE:
```
<tag>content</tag>
<TLMSP>A context aware security protocol</TLMSP>
<br/>
```

Attributes are name-value pairs which decorate start-tags or empty-tags.

7.7.2 Using XML with TLMSP

EXAMPLE 1:
```
<credentials x-context=2>
   <user nationality="SE" encoding="UTF-8">Sven</user>
   <password x-context=3>secret</password>
</credentials>
```

EXAMPLE 2: Example 1 with context indicators
```
c1(<credentials x-context=2>)c2(
   <user nationality="SE" encoding="UTF-8">Sven</user>
   <password x-context=3>)c3(secret)c2(</password>)
c1(</credentials>)
```

There could be an additional attribute which tells if the start-tag and end-tag is supposed to be included in the context specified by the context attribute.

EXAMPLE 3:
```
<br/ x-context=2 x-contextTag="true">
```

7.8 HTML

7.8.1 Introduction

HTML is made for browsing the internet. It is very like XML in its structure. Typically, HTML content is provided within a HTTP body. It generally starts with some metadata and predefinitions, followed by a HTML body enclosed by a body tag. Headings are indicated with h tags, h1, h2, etc., where the number resembles heading 1, heading 2, etc. A simple webpage has headings and paragraphs, enclosed in p tags. The attributes style and class on the tags make it possible to change the layout, like font, font size, colouring, etc.

EXAMPLE 1:
```
<body style="background-color:red;">
<h1 class="h1class">Heading</h1>
<p>An example</p>
</body>
```

A class is defined within a style tag.

EXAMPLE 2:
```
<style>
  .abc {
    background-color:green;
  }
</style>
<div class="abc">
…
</div>
```

A division in HTML, typically gives a layout of the page and the content is enclosed by a `div` tag. Smaller parts can be enclosed in `span` tags, which also have style and class attributes.

Forms in HTML, where a user is able to fill in content or upload files, are coded using the FORM tag. An important example of content of the form tag is input tags, which with an attribute named "`text`" gives the possibility to edit text to be sent using the button constructed with the attribute "`submit`". The form tag has an action attribute containing the location to where the form data is supposed to be sent, and a method attribute describing how it is sent. For instance, by a `GET` or `POST` HTTP request. The actual form data is then coded in HTTP.

7.8.2 Feasibility of using HTML with TLMSP

The approach from XML ought to be possible to use. In the style and class attributes, it would be possible to add some context to it. Classes could be linked in from stylesheets in separate css-files, which could be located on a completely different server. In a context aware setting it would be difficult for a server to keep track of external stylesheets, and the recommendation ought to be to only place TLMSP context definitions together with the actual HTML content.

7.8.3 Using HTML with TLMSP

An example illustrating the use of an `x-context` attribute of a class definition within a style tag. The actual content of the div tag with this class attribute would then be sent in that particular context.

EXAMPLE 1:
```
<style>
  .abc {
    background-color:blue;
    x-context:2;
  }
</style>
<div class="abc">
…
</div>
```

If an `input` tag in a form has an `x-context` attribute, typically, the client would send the content of the input tag within a container of the specified context. See also clause 7.3.

EXAMPLE 2:
```
<form action="/induce" method="get">
<input type="text" id="user" name="username" x-context="1">
<input type="text" id="pwd" name="password" class="abc">
<input type="submit" value="Submit">
…
</form>
```

7.9 SIP

SIP [i.12] is a very widely used protocol to initiate IP multimedia sessions like VoIP or videoconferencing and it consists of exchanging a number of request-response messages.

The use-cases for SIP are typically to set up user-to-user sessions via a SIP-infrastructure (consisting of SIP-proxies). However, except perhaps for some enterprise scenarios, a caller (represented by a SIP user agent) typically only establishes a connection to its own local service provider/operator, and from there, the SIP infrastructure internally takes care of reachability and establishes the connection to the service provider of the callee, and from there to the callee itself. Therefore, the usefulness of TLMSP "end-to-end" is perhaps not so clear cut.

However, the syntax if SIP is essentially identical to that of HTTP/1.1 as described in clause 7.3.1, i.e. consisting of a start-line, followed by some headers and a (possibly empty) message body. Thus, the discussion how to map these message components to TLMSP contexts applies here too, e.g. map caller information (caller and callee identities in "`To:`" and "`From:`" headers) to contexts with appropriate protection/access rights.

One interesting potential use is the following. The body of a SIP `INVITE` message, trying to set up a call between Alice and Bob say, contains a session description part [i.10] where parameters of the media session is provided, e.g. which media CODECs and security settings that are being offered by Alice [i.11]. Crypto attributes such as encryption keys (that would otherwise be carried in the clear) would obviously benefit from being associated with and end-to-end secured context in those cases where TLMSP can be used.

8 Application development with TLMSP

8.1 TLMSP user interface

This clause contains a sketch of how the user-interface for a TLMSP implementation might look like. To be as general as possible, rather than focusing on a specific application, the use-case considered is a generic one where TLMSP is integrated into a browser. To provide a setting of some familiarity to many readers, the user interface of the Mozilla Firefox browser has been used as a template.

The term "User interface" is here to be understood as the configuration and management interface as part of the general browser settings menu. As will be seen, usage of TLMSP might also result in various user dialogues appearing, e.g. asking the user to explicitly authorize certain events. It is envisioned that these can be handled in a fairly standard way, prompting the user with some text describing the type of authorization sought, and a pair of "Accept" and "Deny" buttons. This type of dialogue is therefore omitted.

First, as the user selects the "Settings" item in the browser menu, there is typically a "Security and Privacy" tab where currently HTTP-settings can be managed. It is envisioned that this is also a suitable location for TLMSP-settings. Thus, when the user has selected this tab, a screen somewhat like Figure 21 will appear.

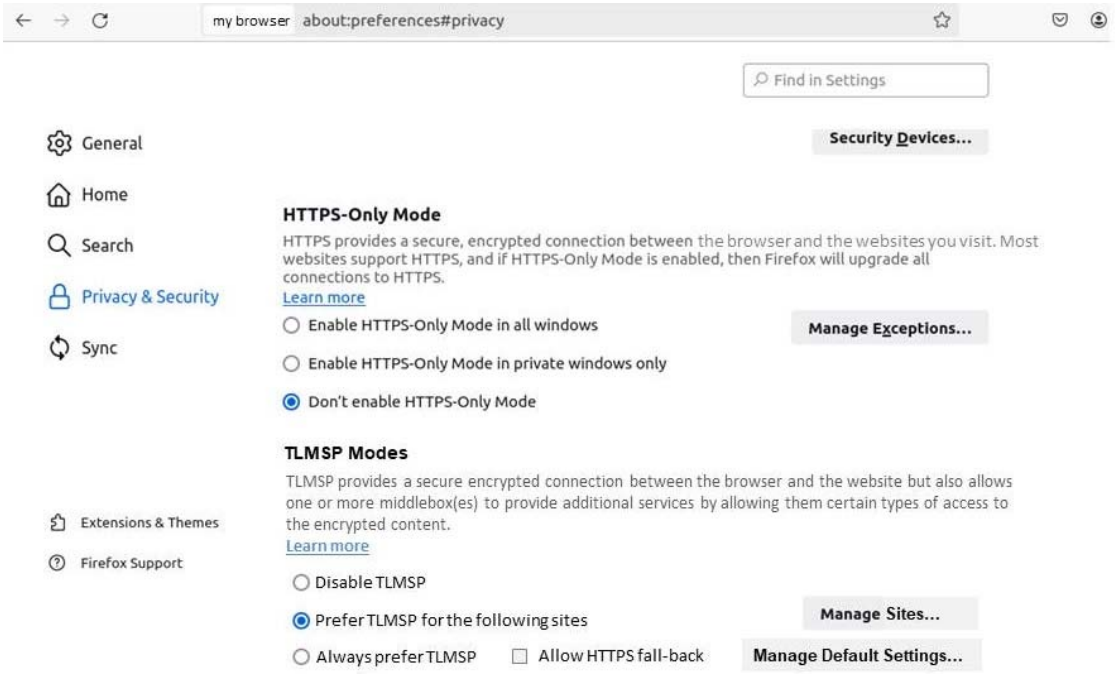


Figure 21: TLMSP-settings under "Security and Privacy

The user can here choose whether to completely disable TLMSP, whether to prefer it (over HTTPS) for certain sites where the user can specify and manage those sites via the "Manage Sites…" button. The user can also choose to always prefer TLMSP, in which case the user also specifies whether he/she accepts TLS/HTTPS fall-back (for sites that do not support TLMSP). The user can also define generic setting in this case, e.g. which middleboxes that are requested, what contexts to use, etc. The details of that are quite similar for the case of a specific site, which is the focus of the rest of this clause.

Thus, assuming the user next wants to define the sites for which TLMSP is preferred, the user clicks "Manage Sites…". This leads to the following input dialogue.

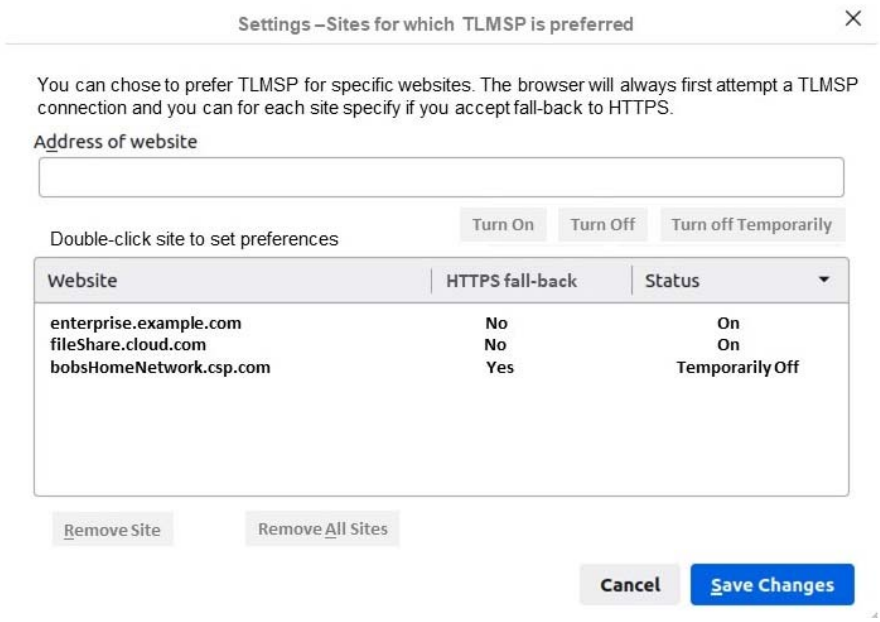


Figure 22: TLMSP-settings after choosing "Manage Sites…"

Here, the user enters URLs to those sites for which TLMSP is preferred, and the selected sites appear in the list. In this case, the user has entered three sites. The last site, bobsHomeNetwork.csp.com corresponds to access to some "IoT gadgets" in Bob's house. Bob is currently trouble-shooting his home network and has therefore temporarily disabled TLMSP for that URL to rule out the possibility that it is TLMSP and/or the middleboxes for that connection that causes problems. The actual settings for each are reached by double-clicking the corresponding URL.

Next, by double-clicking "enterprise.example.com", corresponding to Bob's remote access into the corporate network where Bob works, the configuration option of that specific TLMSP-connection shows as in Figure 23.

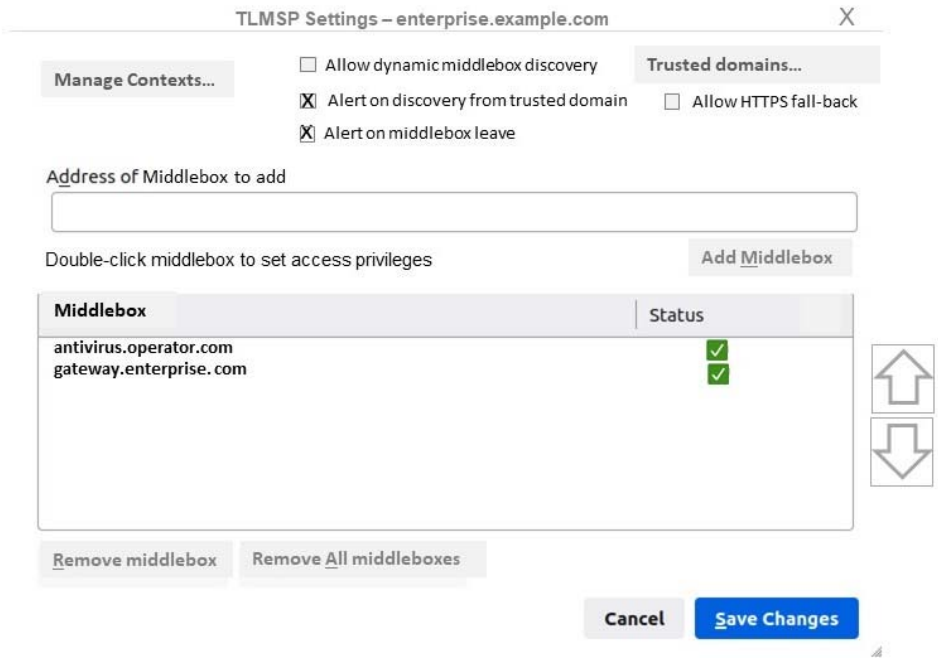


Figure 23: TLMSP-settings after choosing "Manage Sites…"

Here, Bob can by clicking "Manage Contexts…" define which contexts that are desired for this connection (more on this below). Bob can also specify:

- whether he accepts dynamically discovered middleboxes (ones that are not pre-configured in the middlebox list);
- a specific set of domains such that middleboxes from those domains are considered particularly trustworthy for dynamic discovery;
- whether or not he wants to explicitly authorize dynamic discovery from those trusted domains (discovery from other domains are assumed to always require explicit authorization);
- whether HTTPS fall-back is accepted for this connection; and
- whether there will be alerts when a middlebox leaves the session.

There are two arrow-buttons to the right which enables Bob to specify and change the order in which middleboxes are to participate in the session.

In this case, Bob is assumed to have an established session with enterprise.example.com, and the green "check-boxes" indicate that both middleboxes are authenticated and present in the session.

In case middlebox discovery occurs during TLMSP handshake Bob might be shown a dialogue like the one shown in Figure 24.

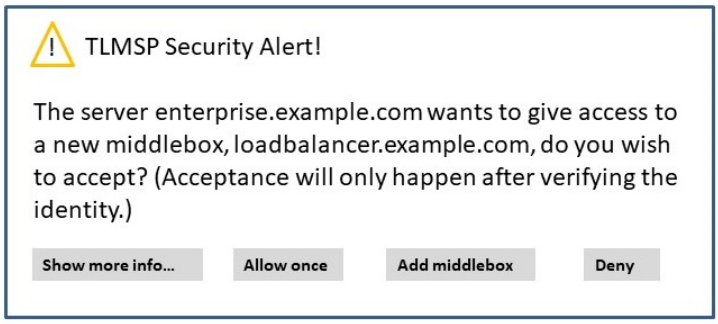


Figure 24: Bob is alerted about dynamic middlebox discovery

By clicking "Show more info…", Bob could manually inspect further details such as the certificate of the proposed middlebox, the reason for adding the middlebox (the TLMSP middlebox list extension can encode the intended purpose of each middlebox), what access rights is intended, etc. Bob can chose to accept or deny the middlebox already at this point. However, since this dialogue takes place when receiving the ServerHello, i.e. before the certificate and middlebox authenticity has be cryptographically verified, Bob is also informed that such verification is still pending and could thus result in rejecting the middlebox even if Bob finds it acceptable as such.

Returning to Figure 23, in order to configure the middlebox access privileges, Bob can double-click the corresponding middlebox URL and is then presented with the following input dialogue shown in Figure 25.

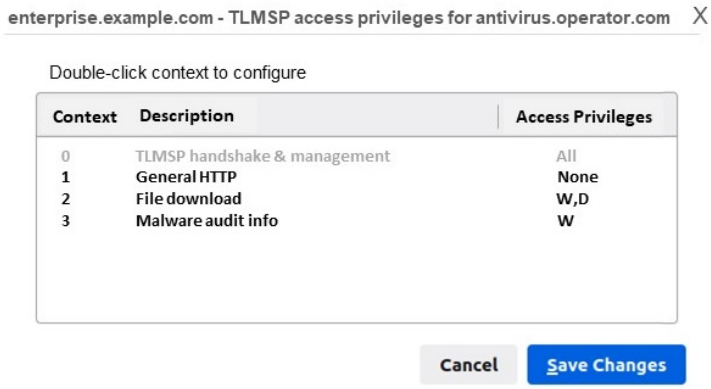


Figure 25: Middelbox-specific configuration

As can be seen, there is a total of four contexts in the session, numbered 0 to 3, where context 0 is the always-present context used for handshake and general TLMSP control. By right-clicking a context in the "Access privilege" column, Bob can change the access privileges to that context for this middlebox. Context 0 is "greyed out" since the settings for this context cannot be modified (all middleboxes always have full access to context 0, indicated by "All" in the rightmost column). Context 1 is used for end-to-end security between the client and the server, thus the middlebox has no privilege associated with this context (indicated by "None"). Contexts 2 and 3 are more interesting. Context 2 is used for file downloads within the session, and the middlebox in question, "antivirus.operator.com" has been added for malware-removal purposes. Thus, this middlebox requires Delete (D) privilege for this context, and to ensure the application does not crash due to removed files, the middlebox also has write (w) privilege, enabling inserts. Further, it is desirable for the middlebox to provide the user with information about removed malware. To that end, context 3 is used, where the middlebox is authorized to insert (write) messages in the form of audit containers, and Bob has also requested that the content of these audit containers is shown to him (e.g. "Malware type bad_code_4711 detected and removed").

Finally, to manage the contexts, Bob can as mentioned click on "Manage Contexts…" shown in Figure 26. Bob is then first shown a list of contexts with the possibility to add/remove contexts (except for context 0, which cannot be removed according to ETSI TS 103 523-2 [i.6]). As discussed in clause 7, there needs to be a way to map parts of the application (in this case HTTP) to contexts. This can be done by either selecting a browser parser-plug-in, or, to use a file which defines these mappings, i.e. how to parse application layer messages. In this case, Bob has chosen the file "enterprise_TLMSP_policy" on his local computer.

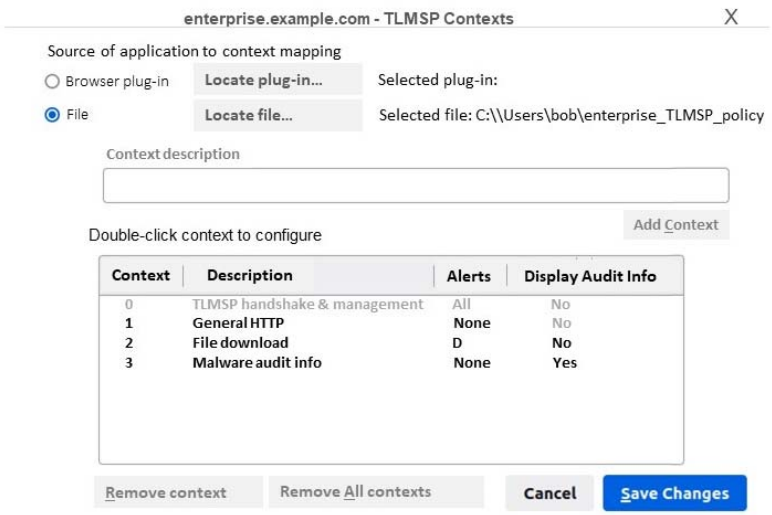


Figure 26: TLMSP context management settings

Bob can also define new contexts, assigning them a descriptive name. Further, by double-clicking a context, Bob can specify whether he wants to display alerts and/or audit information associated with a context.

8.2 TLMSP in the OSI model

Like many other real-world protocols, such as TLS, TLMSP does not fit cleanly in the OSI model. However, this clause provides a brief discussion and reasoning how TLMSP might be described from the OSI model standpoint.

The OSI-model [i.15] is a well-known standardized model serving as a basis for standards development for networking. It defines a model comprising seven different layers, each layer providing service to the layer above, and utilizing the layer below. It can be used to create "clean" standards for a certain layer, knowing which types of service it might need to support as well as what types of services can be used. It is often a goal to avoid so called layer violations where a certain standard cannot be uniquely associated with a specific layer.

In regards to security services, functions such as encryption are typically placed at the presentation layer. There are however no rules without exception, and sometimes it might be more advantageous to place security at a higher layer (since it allows inspection by lower layers) while in other situations, security is preferably placed a lower layer (since it protects more of the data, such as meta-data from higher layers).

Due to the close relationship between TLMSP and TLS, much of what applies to TLS also applies to TLMSP:

- TLMSP uses a transport protocol to send and receive messages to/from its peers in a hop-by-hop topology. This places TLMSP above the transport layer.
- Some of the TLMSP messages establishes sessions with its peers. This places TLMSP in the session layer.
- Application layer protocols such as HTTP use TLMSP to set access rights for middle-boxes and to send messages to other applications. This places TLMSP below the application layer.

So, in conclusion TLMSP resides between the application layer and the transport layer, i.e. in the presentation and session layers.

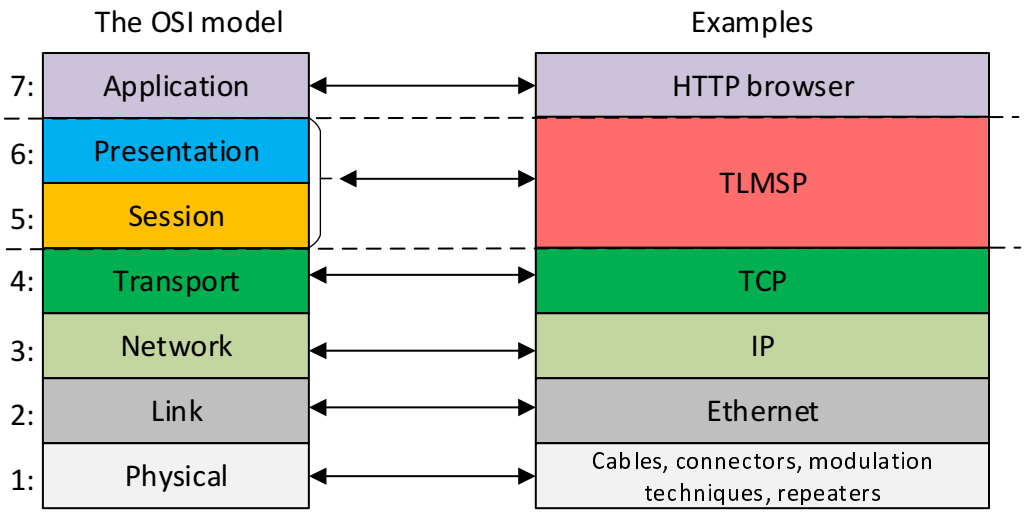


Figure 27: TLMSP covers the presentation layer and the session layers in the OSI model

Is TLMSP different from TLS? The main difference compared to TLS is that TLMSP allows intermediaries to "interfere" with the transport of application layer data via "write", "delete" and "insert" operations at a middlebox. Conceptually, the "delete" capability causes no extra issues since TLS would also drop messages if message integrity check fails. The "write/insert" capability is intended to support the application in ways that TLS cannot achieve. Obviously, to utilize these features, application-specific APIs need to be implemented as discussed in clause 8.3. However, such an API fits well with TLMSP being considered at the presentation layer, just below the application layer. It just seems to require a more sophisticated API (and possibly user interface, see clause 8.1) than the one in which putting an "s" after "http" triggers a TLS handshake. Another difference is the TLMSP feature of middlebox discovery, but this does not seem to conceptually change the location within the stack. Thus, the conclusion is that TLMSP can be mapped to the OSI model in analogy with TLS.

8.3 Application Programming Interface (API)

8.3.1 Overview

This clause and annex A describes a possible functional API between a TLMSP capable application and an underlying TLMSP library. A functional API consists of a number of from the application layer callable functions carrying zero or more input parameters. The functions return zero or more output parameters after execution. See Figure 28.

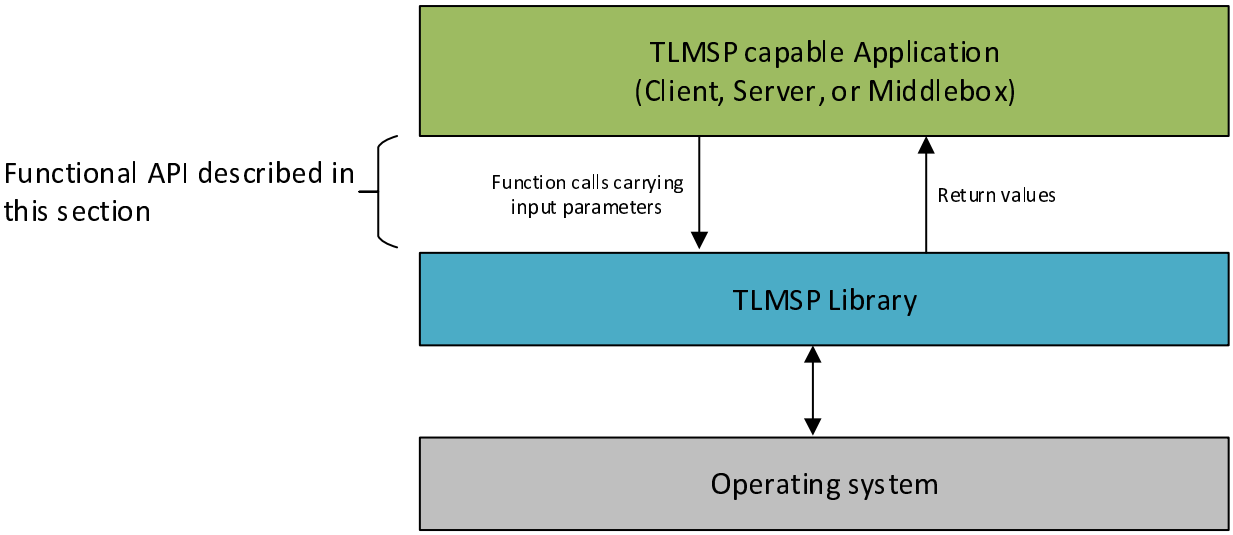


Figure 28: A functional API between a TLMSP application and TLMSP library

The TLMSP library itself typically resides on top of the operating system and uses various system calls to provide its services to the application. The TLMSP library implements (among other things) encryption, key generation, and certificate handling, or has dependencies to other libraries that implement these functionalities.

Annex A consists of an extensive list of type descriptions and functional descriptions that can be used as reference or inspiration when defining a TLMSP API.

8.3.2 Typical call flow of client connecting to a server

This clause briefly outlines the order in which some of the API functions are called when setting up a simple TLMSP client. The client should first be initialized with contexts, its own client address, the server's address, and a list of middleboxes. This is typically done using the following functions:

- `TLMSP_set_contexts()`
- `TLMSP_set_client_address()`
- `TLMSP_set_server_address()`
- `TLMSP_set_initial_middleboxes()`

The client is now ready to connect using TLMSP_middlebox_handshake(), and if the connection was successfully established, the client can now start reading and writing containers using `TLMSP_container_write()` and `TLMSP_container_read()`.

See annex A for a comprehensive list of functions defining the API and [i.22] for source code implementing the API.

9 TLMSP in the Zero Trust model

9.1 Background

Zero Trust is a cyber-security paradigm standardized in 2020 by NIST [i.4] and mandated for U.S. federal agencies in 2021 by an Executive Order [i.1].

The basic assumptions to reach a Zero Trust Architecture are recorded in seven tenets in the NIST standard. In this clause it is explored how to employ the TLMSP protocol to support a Zero Trust Architecture by describing how the TLMSP middleboxes are useful for each Zero Trust tenet.

9.2 TLMSP in relation to ZT tenets

9.2.1 Tenet 1

Tenet 1 [i.4] states that:

"All data sources and computing services are considered resources. *A network may be composed of multiple classes of devices. A network may also have small footprint devices that send data to aggregators/storage, software as a service (SaaS), systems sending instructions to actuators, and other functions. Also, an enterprise may decide to classify personally owned devices as resources if they can access enterprise-owned resources."* [i.4]

Middleboxes are resources by this definition. Middleboxes can process data and can also be sources of data (if given "write" or "delete" privilege).

9.2.2 Tenet 2

Tenet 2 [i.4] states that:

*"**All communication is secured regardless of network location**. Network location alone does not imply trust. Access requests from assets located on enterprise-owned network infrastructure (e.g. inside a legacy network perimeter) must meet the same security requirements as access requests and communication from any other nonenterprise-owned network. In other words, trust should not be automatically granted based on the device being on enterprise network infrastructure. All communication should be done in the most secure manner available, protect confidentiality and integrity, and provide source authentication."* [i.4].

In a classical enterprise setting a server at a corporate office communicating with a laptop outside might run the traffic unencrypted from the server to the firewall which then turn on TLS. But in a Zero Trust setting the traffic inside the office is typically also to be protected. Creating two separate TLS instances, one within the office and one outside, is not always satisfactory as it breaks the e2e authenticity and the client and the server cannot guard themselves against a man in the middle attack. On the other hand, one TLS instance directly from the server to the client is also not satisfactory, as they might not allow the firewall to remove malware sent from outside. In fact, also for traffic that is entirely assumed network-internal, the fact that the source is assumed internal does not imply that it is trusted. A solution approach is to use TLMSP. A firewall middlebox applying TLMSP will allow the client and server to verify that they are not subjected to a man in the middle attack and allow the firewall to remove malware sent from outside. Obviously, the TLMSP middleboxes need to be made trustworthy: assuming trust just because they are also inside the perimeter does not traverse the issue. This is discussed in relation to tenet 3 below.

9.2.3 Tenet 3

Tenet 3 [i.4] states that:

*"**Access to individual enterprise resources is granted on a per-session basis.** Trust in the requester is evaluated before the access is granted. Access should also be granted with the least privileges needed to complete the task. This could mean only "sometime recently" for this particular transaction and may not occur directly before initiating a session or performing a transaction with a resource. However, authentication and authorization to one resource will not automatically grant access to a different resource."* [i.4].

A classical middlebox could be constructed by simply providing it with all key-material to decrypt and encrypt, providing complete access. That would not be coherent with Zero Trust and least privilege access. TLMSP allows for a fine-grained privilege setup using contexts, differentiating different types of data and if a middlebox has read, delete or write access. The middleboxes in TLMSP do not inherit any long-term privileges, the client and server need to agree on all middleboxes between them at handshake, and at the end of that session those access rights are revoked.

9.2.4 Tenet 4

Tenet 4 [i.4] states that:

*"**Access to resources is determined by dynamic policy - including the observable state of client identity, application/service, and the requesting asset - and may include other behavioural and environmental attributes.** An organization protects resources by defining what resources it has, who its members are (or ability to authenticate users from a federated community), and what access to resources those members need. For zero trust, client identity can include the user account (or service identity) and any associated attributes assigned by the enterprise to that account or artifacts to authenticate automated tasks. Requesting asset state can include device characteristics such as software versions installed, network location, time/date of request, previously observed behaviour, and installed credentials. Behavioural attributes include, but are not limited to, automated subject analytics, device analytics, and measured deviations from observed usage patterns. Policy is the set of access rules based on attributes that an organization assigns to a subject, data asset, or application. Environmental attributes may include such factors as requestor network location, time, reported active attacks, etc. These rules and attributes are based on the needs of the business process and acceptable level of risk. Resource access and action permission policies can vary based on the sensitivity of the resource/data. Least privilege principles are applied to restrict both visibility and accessibility."* [i.4].

There are two different situations at hand. Firstly, how middleboxes can change access to other resources, and secondly, how access to middleboxes as a resource can be changed.
The first situation. The enforcement of dynamic access policies is at the core of Zero Trust. Middleboxes are designed to perform those duties. One could for example have one middlebox between the client and server specialized in the dynamic policies generated by an AI/ML device trained on a particular type of data; and then another separate middlebox using information entered by a network technician based on intelligence gathering. The TLMSP protocol allows the server to guarantee that those middleboxes are situated between it and the client.

The middleboxes themselves are considered as a resource and could thus also be compromised. This is the same risk as any firewall with keys to decrypt traffic is subjected to, and the risk should be mitigated. Locating this risk at the middleboxes instead of at more extensive proprietary enterprise resources allows for a selective high assurance methodology, which could include a formal code verification of the middleboxes. According to Gligor et al [i.2], focusing strong verification to a small number of high-assurance components optimizes the cyber security/costs trade-off for the complete enterprise. As the middleboxes between the client and server are negotiated at handshake and not permanently fixed, any sign of security problems could be initially addressed by e.g. using remote attestation and excluding the vulnerable middlebox and replacing it. If the security issues are vague or uncertain, the granularity of the middlebox settings in the TLMSP protocol also allow for restricting its rights, for example downgrading from write to read if the security issue is marginal and the functionality of the middlebox important.

9.2.5 Tenet 5

Tenet 5 [i.4] states that:

*"**The enterprise monitors and measures the integrity and security posture of all owned and associated assets.** No asset is inherently trusted. The enterprise evaluates the security posture of the asset when evaluating a resource request. An enterprise implementing a ZTA should establish a Continuous Diagnostics and Mitigation (CDM) or similar system to monitor the state of devices and applications and should apply patches/fixes as needed. Assets that are discovered to be subverted, have known vulnerabilities, and/or are not managed by the enterprise may be treated differently (including denial of all connections to enterprise resources) than devices owned by or associated with the enterprise that are deemed to be in their most secure state. This may also apply to associated devices (e.g. personally owned devices) that may be allowed to access some resources but not others. This, too, requires a robust monitoring and reporting system in place to provide actionable data about the current state of enterprise resources."* [i.4].

The middleboxes in TLMSP are ideal for collecting and processing information to the CDM system. Clients or servers that are vulnerable or subverted will show signs of that in the traffic between them, allowing for middleboxes to find those signs and report it to the CDM system. In particular, this might apply to personally owned devices that are traditionally considered outside the scope of security considerations but are within according to Zero Trust tenets.

9.2.6 Tenet 6

Tenet 6 [i.4] states that:

*"**All resource authentication and authorization are dynamic and strictly enforced before access is allowed.** This is a constant cycle of obtaining access, scanning and assessing threats, adapting, and continually reevaluating trust in ongoing communication. An enterprise implementing a ZTA would be expected to have Identity, Credential, and Access Management (ICAM) and asset management systems in place. This includes the use of multifactor authentication (MFA) for access to some or all enterprise resources. Continual monitoring with possible reauthentication and reauthorization occurs throughout user transactions, as defined and enforced by policy (e.g. time-based, new resource requested, resource modification, anomalous subject activity detected) that strives to achieve a balance of security, availability, usability, and cost-efficiency."* [i.4].

Middleboxes can be employed to each check that different parts of a multifactor authentication has been performed, for example, by allowing the client to present different credentials (certificates) to different middleboxes. A client and server are negotiating the middleboxes on the path in TLMSP, for example the client could propose middleboxes that are able to verify its authentication, and the server could select dynamically which one of them that are necessary, or simply deny access as not enough of relevant ones are included. The server can also propose middleboxes to the client, who then could authenticate itself in the corresponding way and gain access to resources on the server. If it turns out that a certain authentication procedure is compromised, the middleboxes associated to that one could simply be turned off, instead of updating lots of servers and clients on that.

9.2.7 Tenet 7

Tenet 7 [i.4] states that:

"The enterprise collects as much information as possible about the current state of assets, network infrastructure and communications and uses it to improve its security posture. *An enterprise should collect data about asset security posture, network traffic and access Senforcement. This data can also be used to provide context for access requests from subjects."* [i.4].

TLMSP is suitable for implementing this Zero Trust tenet. To gather security information regarding TLS sessions between Alice and Bob would before TLMSP require that at least one of them compile and submits it to the Continuous Diagnostics and Mitigation (CDM) system. But Alice or Bob might constitute security problems themselves, not to dwell on the implementational issues of submitting standardized data to the CDM. This is all resolved by inserting a TLMSP middlebox between Alice and Bob that is designed to submit data to the CDM according to its dynamically assessed current needs.

9.3 Migration to ZT

The implementation of Zero Trust does not usually start from the technology agnostic seven tenets and right off to software development; a more detailed document for understanding the enterprise situation of the company or agency is called for. To support U.S. Federal Civilian Executive Branch agencies in that, the Cybersecurity and Infrastructure Security Agency published its Zero Trust Maturity Model (ZTMM). In ZTMM the tenets are boiled down to action points for the five pillars: Identity, Devices, Networks, Applications & Workloads and Data. For each pillar the essential functions are described and their road to ZTMM. For example, in the Network pillar, the road towards ZTMM for the Network Segmentation function is:

Traditional: The enterprise defines their network architecture using large perimeter/macro- segmentation with minimal restrictions on reachability within network segments. The enterprise might also rely on multi-service interconnections (e.g. bulk traffic VPN tunnels).

Initial: The enterprise begins to deploy network architecture with the isolation of critical workloads, constraining connectivity to least function principles, and a transition toward service-specific interconnections.

Advanced: The enterprise expands deployment of endpoint and application profile isolation mechanisms to more of their network architecture with ingress/egress micro- perimeters and service- specific interconnections.

Optimal: The enterprise network architecture consists of fully distributed ingress/egress micro- perimeters and extensive micro-segmentation based around application profiles with dynamic just-in-time and just-enough connectivity for service-specific interconnections. [i.5].

The TLMSP middleboxes are useful tools to achieve this ZTMM functionality. Replacing the big parameter with TLMSP connections allows for dynamic and extensive micro-segmentation controlled by them. At the start of the path those middleboxes have options to simply do nothing, simply falling back on traditional TLS, and then gradually down the road towards optimal Zero Trust, more and more dynamic control is introduced, towards reaching the optimal goal.

For the Network pillar, the remaining optimal ZTMM goals are:

1) **Traffic Encryption:** The enterprise continues to encrypt traffic as appropriate, enforces least privilege principles for secure key management enterprise- wide, and incorporates best practices for cryptographic agility as widely as possible.

2) **Network Resilience:** The enterprise integrates holistic delivery and awareness in adapting to changes in availability demands for all workloads and provides proportionate resilience.

3) **Visibility and Analytics Capability:** The enterprise maintains visibility into communication across all agency networks and environments while enabling enterprise-wide situational awareness and advanced monitoring capabilities that automate telemetry correlation across all detection sources.

4) **Automation and Orchestration Capability:** The enterprise networks and environments are defined using infrastructure-as-code managed by automated change management methods, including automated initiation and expiration to align with changing needs.

5) **Governance Capability:** The enterprise implements enterprise-wide network policies that enable tailored, local controls; dynamic updates; and secure external connections based on application and user workflows [i.5].

To reach optimal functionality, the TLMSP protocol can be a very suitable tool. The middleboxes provide the backbone to enforce all of the introduced dynamic and holistic network security measures, and reduces costs as security is concentrated to the new middleboxes instead of proprietary or legacy software designed long ago. Note in particular the encryption and visibility aspects (items 1 and 3) which, without solutions such as TLMSP, remains as conflicting goals.

10 Summary and conclusions

The present document has had the goal to show how to use TLMSP in a number of common and widely applicable use cases and how TLMSP could be integrated into these use cases. While it cannot be denied that a main purpose has been to show advantages of TLMSP compared to other solutions, it has at the same time been a goal to stay honest about some limitations of TLMSP by stressing them and giving the implementor useful insight when a particular middlebox protocol is to be chosen. It is clear that TLMSP is quite powerful protocol for middlebox-solutions, but it comes at a cost in terms of more complex configuration needs.

Guidelines on how to integrate TLMSP in the communication stack, with a particular focus on user interface has also been presented.

Annex A:
Application Programming Interface (API)

A.1 Type descriptions

This clause briefly describes the different datatypes used in the API:

- **TLMSP_Context:** A TLMSP context according to clause 4.2.1.2 in ETSI TS 103 523-2 [i.6], i.e. a set of parameters defining the protection of some portions of information.
- **TLMSP_Contexts:** A list of one or more TLMSP_Contexts.
- **TLMSP_Middlebox:** Represents a specific TLMSP middlebox and contains parameters associated with a particular middlebox.
- **TLMSP_Middleboxes:** A list of TLMSP middleboxes.
- **TLMSP_Container:** A TLMSP container according to clause 4.2.1.2 in ETSI TS 103 523-2 [i.6].
- **TLMSP_Discovery_Cb_Function:** Callback function called when new middleboxes are discovered.
- **TLMSP_Context_Auth:** Describes the access rights of a TLMSP context, i.e. one of "None", "Read", "Write", or "Delete".
- **TLMSP_Connection_Ctx:** Various configuration and data relevant to TLMSP session establishment
- **TLMSP_Connection:** An active TLMSP connection between two nodes, i.e. the connection to the "next hop" in the middlebox chain.
- **TLMSP_Address:** The network address of a TLMSP entity, which is either a client, middlebox, or a server.

A.2 Functional descriptions

A.2.1 Introduction

This list of functions are the most prominent ones and constitutes the base of the API. Other functions could very likely have to complement this list in a fully-fledged implementation. Unless otherwise noted, the API is relevant for all entities (i.e. all of server, client and middleboxes).

A.2.2 Managing connections

This clause describes various functions used for connection management.

TLMSP_set_contexts(TLMSP_Connection_Ctx, TLMSP_Contexts)

Initialize a connection context with one or more TLMSP contexts. This comprises all contexts relevant on this connection.

Input

- **TLMSP_Connection_Ctx:** A connection context.
- **TLMSP_Contexts:** A list of TLMSP contexts.

Output

- Returns 1 on success, or 0 on failure.

TLMSP_set_initial_middleboxes(TLMSP_Connection_Ctx, TLMSP_Middleboxes)

Initialize a connection context with one or more initial middle boxes. These are middleboxes that are known at the time when a connection is established. This list of middleboxes is used in the initial ClientHello message.

Input

- **TLMSP_Connection_Ctx:** A connection context.
- **TLMSP_Middleboxes:** A list of TLMSP middleboxes.

Output

- Returns 1 on success, or 0 on failure

TLMSP_set_transparent(TLMSP_Connection_Ctx, TLMSP_Address)

Set the middlebox mode in the connection context to Transparent and set the address that will be inserted into ClientHello messages.

Input

- **TLMSP_Connection_Ctx:** A connection context.
- **TLMSP_Address:** Middlebox address.

Output

- Returns 1 on success, or 0 on failure.

TLMSP_set_discovery_cb(TLMSP_Connection_Ctx, TLMSP_Discovery_Cb_Function)

Register a callback function that will be called by the library when new middleboxes are discovered on this connection.

Input

- **TLMSP_Context:** A specific TLMSP context.
- **TLMSP_Discovery_Cb_Function:** Function to call when discovery is triggered.

Output

- Returns 1 on success, or 0 on failure.

TLMSP_set_client_address(TLMSP_Connection_Ctx, TLMSP_Address)

Sets client address for a specific connection context.

Input

- **TLMSP_Connection_Ctx:** The specific TLMSP connection context.
- **TLMSP_Address:** The client address.

Output

- Returns 1 on success, or 0 on failure.

TLMSP_get_client_address(TLMSP_Connection_Ctx)

Return the client address for a specific connection context.

Input

- **TLMSP_Connection_Ctx:** The specific TLMSP connection context.

Output

- **TLMSP_Address:** The client address of this connection context.

TLMSP_set_server_address(TLMSP_Context_Ctx, TLMSP_Address)

Sets server address for a specific connection context.

Input

- **TLMSP_Connection_Ctx:** The specific TLMSP connection context.
- **TLMSP_Address:** The server address.

Output

- Returns 1 on success, or 0 on failure

TLMSP_get_server_address(TLMSP_Context_Ctx)

Return the client address for a specific connection context.

Input

- **TLMSP_Connection_Ctx:** The specific TLMSP connection context.

Output

- **TLMSP_Address:** The server address of this connection context.

TLMSP_get_first_hop_address(TLMSP_Connection_Ctx)

This function is only relevant for clients and returns the address of any middleboxes between the client and the server. The server address is returned in case there are no middleboxes in the chain.

Input

- **TLMSP_Connection_Ctx:** The specific TLMSP connection context.

Output

- **TLMSP_Address**: The first hop address, i.e. the first middlebox in the chain, or the server if there are no middleboxes.

TLMSP_get_next_hop_address(TLMSP_Connection)

Returns the next-hop address in the chain. This function is typically called by a middlebox in order to figure out the address to the next node to connect to. This information is present in the ClientHello message.

Input

- **TLMSP_Connection:** The specific TLMSP connection.

Output

- **TLMSP_Address:** The next hop address, i.e. the next middlebox in the chain, or the server if there are no middleboxes.

TLMSP_context_access_add(TLMSP_Context, TLMSP_Context_Auth)

Add access rights to a specific TLMSP context.

Input

- **TLMSP_Context:** The specific TLMSP Context.
- **TLMSP_Context_Auth:** Access rights that are to be added to this context.

Output

- Returns 1 on success, or 0 on failure.

TLMSP_context_access_free(TLMSP_Context)

Free access rights to a specific TLMSP context.

Input

- **TLMSP_Context:** The specific TLMSP Context.
- **TLMSP_Context_Auth:** Access rights that are to be added to this context.

Output

- Returns 1 on success, or 0 on failure.

A.2.3 Connection establishment or Managing connections

TLMSP_middlebox_handshake(TLMSP_Connection_Ctx, TLMSP_Connection_Ctx)

Initiate a handshake between two connection contexts.

Input

- **TLMSP_Connection_Ctx:** The first connection context (i.e. the entity that initiates the connection).
- **TLMSP_Connection_Ctx:** The second connection context (i.e. the entity that responds to the connection request).

Output

- Returns 1 on success, or 0 on failure.

A.2.4 Client and server read/write

TLMSP_set_current_context(TLMSP_Connection, TLMSP_Context)

Associate a connection with a specific TLMSP context.

Input

- **TLMSP_Connection:** A TLMSP connection.
- **TLMSP_Contex:** The specific TLMSP context.

Output

- Returns 1 on success, or 0 on failure.

TLMSP_get_current_context(TLMSP_Connection)

Get the TLMSP context associated with a specific connection.

Input

- **TLMSP_Connection:** A TLMSP connection.

Output

- The TLMSP context associated with this connection.

TLMSP_get_last_read_context(TLMSP_Connection, TLMSP_Context)

Get the TLMSP context associated with a specific connection that was last read using TLMSP_Container_Read ().

Input

- **TLMSP_Connection:** A TLMSP connection.

Output

- The TLMSP context associated with this connection.

A.2.5 Container access

TLMSP_container_read(TLMSP_Connection, TLMSP_Container)

Reads data from a connection and put it in a container.

Input

- **TLMSP_Connection:** A TLMSP connection.
- **TLMSP_Container:** A reference to a TLMSP container.

Output

- The TLMSP container with read data.
- Returns 1 on success, or 0 on failure.

TLMSP_container_write(TLMSP_Connection, TLMSP_Container)

Writes from a container to a connection.

Input

- **TLMSP_Connection:** A TLMSP connection.
- **TLMSP_ Container:** A TLMSP container.

Output

- The TLMSP container with data to be written.
- Returns 1 on success, or 0 on failure.

TLMSP_container_delete(TLMSP_Connection, TLMSP_Container)

Deletes plaintext from the container and marks it as deleted.

Input

- **TLMSP_Connection:** A TLMSP connection.
- **TLMSP_Container:** A reference to a TLMSP container.

Output

- The TLMSP container with data.
- Returns 1 on success, or 0 on failure.

TLMSP_container_create(TLMSP_Connection, TLMSP_Container, TLMSP_Context, void*, Size)

Creates and initializes a TLMSP container.

Input

- **TLMSP_Connection:** A TLMSP connection.
- **TLMSP_Container:** A reference to a TLMSP container.
- **TLMSP_Context:** A TLMSP context.
- **Data:** A reference to memory containing plain-text data.
- **Size:** The length (in bytes) of the memory containing plain-text data.

Output

- An initialized TLMSP container.
- Returns 1 on success, or 0 on failure.

TLMSP_container_create_alert(TLMSP_Connection, TLMSP_Container, TLMSP_Context)

Creates and initializes a TLMSP container for alerts according to clause 4.2.3.1.6 in ETSI TS 103 523-2 [i.6], i.e. containers specifically used for sending alerts.

Input

- **TLMSP_Connection:** A TLMSP connection.
- **TLMSP_Container:** A reference to a TLMSP container.
- **TLMSP_Context:** A TLMSP context.

Output

- An initialized TLMSP container.
- Returns 1 on success, or 0 on failure.

TLMSP_container_free(TLMSP_Connection, TLMSP_Container)

Frees memory used by the container and returns it to the system.

Input

- **TLMSP_Connection:** A TLMSP connection.
- **TLMSP_Container:** A reference to a TLMSP container.

Output

- Returns 1 on success, or 0 on failure.

TLMSP_container_context(TLMSP_Container)

Returns the TLMSP context associated with the container.

Input

- **TLMSP_Container:** A TLMSP container.

Output

- Returns the TLMSP context associated with the container.

TLMSP_container_length(TLMSP_Container)

Returns the length (in bytes) of the TLMSP container.

Input

- **TLMSP_Container:** A TLMSP container.

Output

- The length of the TLMSP container.

TLMSP_container_alert(TLMSP_Container)

Returns the alert data from a specific TLMSP alert container.

Input

- **TLMSP_Container:** A TLMSP alert container.

Output

- The alert data from the container.

TLMSP_container_deleted(TLMSP_Container)

Returns 1 if the container has been deleted, and 0 otherwise.

Input

- **TLMSP_Container:** A TLMSP container.

Output

- Returns 1 if the container has been deleted and 0 otherwise.

TLMSP_container_readable(TLMSP_Container)

Returns 1 if the container is readable, and 0 otherwise. A container is readable when it has been received and not having its status set to "No Access".

Input

- **TLMSP_Container:** A TLMSP container.

Output

- Returns 1 if the container is readable, and 0 otherwise.

TLMSP_container_writable(TLMSP_Container)

Returns 1 if the container is writable, and 0 otherwise. A container is writable when it is not an alert container, and not being either a "No Access" container, or a "Read only" container.

Input

- **TLMSP_Container:** A TLMSP container.

Output

- Returns 1 if the container is writable, and 0 otherwise.

TLMSP_container_get_data(TLMSP_Container)

Returns the container plain text data.

Input

- **TLMSP_Container:** A TLMSP container.

Output

- Returns a reference to container plain text data, or a null reference on failure.

TLMSP_container_set_data(TLMSP_Container, Data, Size)

Loads a container with plain text data.

Input

- **TLMSP_Container:** A TLMSP container.
- **Data:** A reference to plain text data.
- **Size:** The size (in bytes) of the data.

Output

- Returns 1 on success, or 0 on failure.

History

Document history		
V1.1.1	July 2024	Publication