methics

MUSAP

MUSAP Specification

Document version 1.4

Change History

Date	Version	Changes
2023-10-17	0.1	Initial version
2023-10-24	0.2	Added Metadata, Import/Export, Security Principles
2023-10-25	0.3	Added missing table
2023-10-25	0.4	Added architecture chapter
2023-10-26	0.5	Added KeySearchReq, SignatureReq and KeyGenReq tables
2023-10-27	0.6	Updated Step-Up authentication and MFA definitions
2023-11-16	0.7	Review
2023-11-23	0.8	Added details after team review
2023-11-23	0.9	Added initial version of the coupling API
2023-12-08	0.95	Error handling
2024-01-18	1.0	Updated MUSAP API
2024-01-25	1.1	Updated Link API
2024-01-25	1.2	Updated Coupling API
2024-03-22	1.3	Updated Key Attestation API
2024-04-03	1.4	Updated key life-cycle and signature descriptions

Table of Contents

1 Overview {Informative}....5
1.1 Terms and Abbreviations....5
1.2 References....6
1.3 Level of Assurances....6
1.4 Introduction....6
2 MUSAP Overview....8
2.1 Multiple SSCDs....8
2.2 Signature Formats....8
2.3 Key Algorithms....8
2.4 Signature Algorithms....10
2.5 MUSAP Extensions....11
2.6 Step-Up Authentication {Optional}....12
2.7 Multi-Factor Authentication {Optional}....12
2.8 Role Based Access Control {Optional}....12
2.9 Testing and Debugging Support {Optional}....13
3 Architecture....14
3.1 Module 1: MUSAP Library....14
3.2 Module 2: MUSAP Link....15
4 KeyURI....17
5 Key Lifecycle....18
5.1 Key Generation....18
5.2 Key Binding....18
6 Signature....20
7 Key and SSCD Metadata....21
7.1 Storage Implementation....21
7.2 Stored Metadata....21
7.2.1 SSCD Metadata....21
7.2.2 Key Metadata....21
7.3 SSCD Types....23
7.4 Import and Export {Optional}....24
7.5 Key Attestation {Optional}....26
8 MUSAP Link Server....28
8.1 Transport Encryption....30
9 MUSAP UI {Informative}....31
10 MUSAP Link Interfaces....32
10.1 Link API....32
10.2 Coupling API....35
11 MUSAP API Description....37
11.1 General API....37
11.2 Key Discovery API....37
11.2.1 Key Search Request....38
11.2.2 SSCD Search Request....39
11.3 Signature API....39
11.3.1 Signature Request....40
11.4 Key Lifecycle API....40
11.4.1 Key Generation Request....41
11.4.2 Key Update Request....41
11.5 Key Binding API....42

11.5.1 Key Binding Request....42
11.6 Import/Export API....43
11.7 SSCD Interface(s)....43
11.8 Coupling API....44
12 Error Handling....46
13 References....47

1 Overview {Informative}

1.1 Terms and Abbreviations

Term	Explanation
End-User Application	An application that uses MUSAP library to produce signatures. For example EUDI wallet.
End-User	
MUSAP API	Native iOS/Android API (swift/java) compiled binaries which provides API functions described in this document
MUSAP Coupling API	HTTP API between MUSAP API on smartphone and MUSAP Link at server side
MUSAP Library	Native iOS/Android library package containing - reference client (End User reference App) - MUSAP API - documentation
MUSAP Link	The server side component to link Web Application with MUSAP on smartphone.
MUSAP Link API	REST API between MUSAP Link and a Web Application
Optional	This feature is optional and every MUSAP implementation may not include the feature.
Relaying Party	Organization which uses asymmetric key for user authentication. The key material related SSCD can be controlled under MUSAP on user’s smartphone.
RFU	This feature is reserved for future use. This indicates that the functionality is documented even if it hasn't been implemented in the reference implementation version.
Web Application	The Relying Party application that calls MUSAP Link to request signatures
Dongle	NFC or USB dongle. For example Yubikey.
TSP	Trust service providers
LoA	

1.2 References

NIST Special Publication 800-130	https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-130.pdf
SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms	https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.2.pdf
RFC 5652 (CMS)	https://datatracker.ietf.org/doc/html/rfc5652
RFC 7515 (JWS)	https://datatracker.ietf.org/doc/html/rfc7515
RFC 8017 (PKCS#1)	https://datatracker.ietf.org/doc/html/rfc8017

1.3 Level of Assurances

Table below lists mapping of LoA according to different standards

Sr. #	SSCDs	EIDAS LoA(s)	ISO 29115 LoA	NIST SP 800-63-3
1	Phone (Android/ iOS) key store	Low	2	IAL/AAL1
2	eIDAS defined Remote Signature	Substantial /High	3, 4	IAL/AAL3
3	Yubikey via NFC	Substantial /High	2/3, 4	IAL/AAL2
4	Local Signature by Mobile ID (UICC/eUICC)	Substantial /High	3, 4	IAL/AAL3

Table 1: Different SSCDs which will be made available for EUDIW to offer two configs

1.4 Introduction

EUDI Wallet security is based on asymmetric keys and qualified signatures and qualified signature creation devices. The need of a common QSCD interface for different Wallet implementations has been identified in many other projects like Enisa, which defines this interface as Secure Component API See figure below.

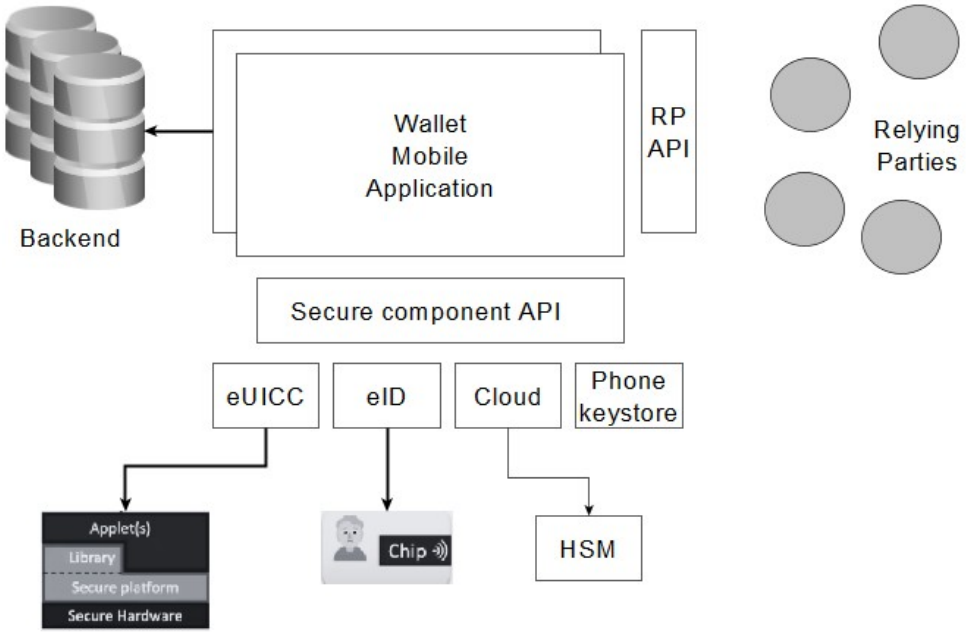

Figure 1: Wallet architecture by Enisa.

The Multiple SSCD with Unified Signature API Library (MUSAP) project develops a new software interface called Unified Signature Application Programming Interface (USAPI) Library for simple and secure app access to eIDAS compliant Type 1 and Type 2 Secure Signature Creation Devices (SSCDs). MUSAP provides in practice the same functionality as Enisa's Secure component API.

MUSAP will act as an intermediary layer between app and SSCD that abstracts the complexities of different SSCD technologies and provides a unified API Library for app developers.
The Framework for Designing Cryptographic Key Management Systems (CKMS) described in the NIST Special Publication 800-130 contains main design topics that have been be considered when developing the MUSAP specification.

The MUSAP project has received funding from the European Union via the TrustChain project. The TrustChain Project is funded by the European Union under GA No 101093274.

2 MUSAP Overview

2.1 Multiple SSCDs

MUSAP API defines key generation and key binding on multiple SSCDs. The main objective of the MUSAP API is to provide users with the choice to select the SSCD implementation that best suits their needs.

Multiple SSCD feature means that the identification of the SSCDs and keys inside must be unified, signature algorithms and signature parameters must be handled explicitly and each SSCD user interaction must be unified as much as possible. Furthermore, the life cycle of keys differs among various SSCD technologies.

To solve these requirements, MUSAP API has been split into several sets of API calls:

1. General — Operations for MUSAP API management
2. Key binding — Operations to bind an existing SSCD and keys with MUSAP
3. Key management — Operations to generate and manage keys.
4. Key discovery — Operations to find a proper key.
5. Import/Export — Operations for exchanging keys between different MUSAP instances.
6. Sign — Operations for signing with SSCD.

2.2 Signature Formats

MUSAP supports at least the following Signature Formats:

Signature Format	Description
PKCS#1	Also sometimes known as RAW. RFC 8017.
CMS	Cryptographic Message Syntax. RFC 5652.
JWS	JSON Web Signature. RFC 7515.

Nevertheless, SSCDs may support also a broader range of formats. New signature formats are supported in MUSAP by requesting them through the Signature API.

2.3 Key Algorithms

MUSAP supports at least the following Key Algorithms. SSCDs may define additional key algorithms. Each of these algorithms might not be supported by every SSCD. The supported

algorithms per SSCD can be checked with the Key Discovery API.

RSA	
Key Algorithm	**Key Size**
RSA2K	2048
RSA4K	4096

ECC		
Key Algorithm	**Key Size (bits)**	**Curve**
ECC P256 K1	256	secp256k1
ECC P256 R1	256	secp256r1
ECC P384 R1	384	secp384r1
Ed25519	256	Curve25519
Brainpool256 R1	256	BrainpoolP256r1
Brainpool384 R1	384	BrainpoolP384r1

2.4 Signature Algorithms

MUSAP supports at least the following Signature Algorithms. SSCDs may define additional signature algorithms. Each of these algorithms might not be supported by every SSCD. The supported algorithms per SSCD can be checked with the Key Discovery API.

Algorithm	Hash Algorithm	Notes
SHA256withECDSA	SHA-256	
SHA384withECDSA	SHA-384	
SHA512withECDSA	SHA-512	
NONEwithECDSA	N/A	Pre-computed hash
SHA256withRSA	SHA-256	
SHA384withRSA	SHA-384	
SHA512withRSA	SHA-512	
NONEwithRSA	N/A	Pre-computed hash
SHA256withRSASSA-PSS	SHA-256	
SHA384withRSASSA-PSS	SHA-384	
SHA512withRSASSA-PSS	SHA-512	
NONEwithRSASSA-PSS	N/A	Pre-computed hash

2.5 MUSAP Extensions

MUSAP allows easily integrating new SSCDs with the SSCD Interface(s). This allows applications using MUSAP to extend their supported SSCDs.

2.6 Step-Up Authentication {Optional}

Step-up authentication, also known as adaptive authentication, is a dynamic authentication process that adjusts the level of authentication required based on specific conditions or risk factors. It doesn't always involve multiple factors; instead, it can escalate the authentication requirements when certain conditions are met.

Step-up authentication is employed when there is a change in the user's behaviour, the sensitivity of the requested action, or a detected risk. For example, it might be used when a user attempts to access a sensitive file, initiate a financial transaction, or logs in from an unfamiliar location.

The purpose of step-up authentication is to provide a flexible and risk-based approach to security. It allows organizations to apply stronger authentication only when necessary, reducing friction for users during routine interactions while adding security when conditions warrant it.

The End-User Application can set a Step-Up Authentication policy for a key which defines when and how MUSAP should ask for additional data from the user. For example, the End-User Application can request user for their biometrics when using the key to sign a transaction. Either key generation, key binding, or signature request can contain the policy. The signature request policy cannot override the other policies to prevent fraud.

The policy has the following parameters:

Parameter	Example Value	Description
Frequency	NEVER	How often MUSAP should the for step-up authentication. Can be always, never, or when requested in a signature request.
Validity duration	10min	How long a step-up authentication remains valid. Can be a duration, number of signatures, or a zero value.
Type	Biometric	What authentication type the user should be provide.

2.7 Multi-Factor Authentication {Optional}

MFA requires users to provide two (two-factor authentication 2FA) or more authentication factors when authenticating. Meanwhile, MUSAP incorporates supplementary SSCDs as additional authentication factors. Consequently, MUSAP has the capability to increase security by combining multiple PINs as distinct authentication factors during the authentication process.

2.8 Role Based Access Control {Optional}

MUSAP has support for simple RBAC by allowing the end-user application to define roles and assign keys to those roles.

Key roles can be defined during key generation via the Key Lifecycle API or during key binding via the Key Binding API. Keys that belong to a specific role can be requested with Key Discovery API.

The goal of the RBAC support is to prevent the user from being presented unwanted keys when a key of a specific role is wanted. This may prevent cases where e.g. a corporate signature was used instead of a personal signature.

2.9 Testing and Debugging Support {Optional}

To support debugging the MUSAP library, the End-User application can enable debug logging on MUSAP library by calling setDebugLog function. Disabling debug logging in production is recommended.

3 Architecture

MUSAP architecture supports both smartphone based apps (local end-user app or eWallet) and web servers (remote web wallets). Smartphone support is defined in Module 1 and web server support is defined in module 2.

3.1 Module 1: MUSAP Library

MUSAP API supports integration of both smartphone based end-user apps (like local eWallet) and web services (remote web wallet).

The MUSAP library consists of the following:

1. Compiled software binaries
2. API documentation
3. Configurations

Compiled software binaries contain the following main MUSAP components:

1. MUSAP API
2. MUSAP Coupling API
3. SSCD Interface(s)
4. Metadata storage

MUSAP API provides a native API for the End-User application to request key operations and interact with the MUSAP library. This API is further divided into parts as shown in the figure below. The API is further explained in the chapter MUSAP API Description.

The SSCD interface provides a way to integrate SSCDs into MUSAP. This component is described in chapter SSCD Interface(s).

The metadata storage is a component that stores and secures key and SSCD metadata. Metadata storage is described in chapter Key and SSCD Metadata.

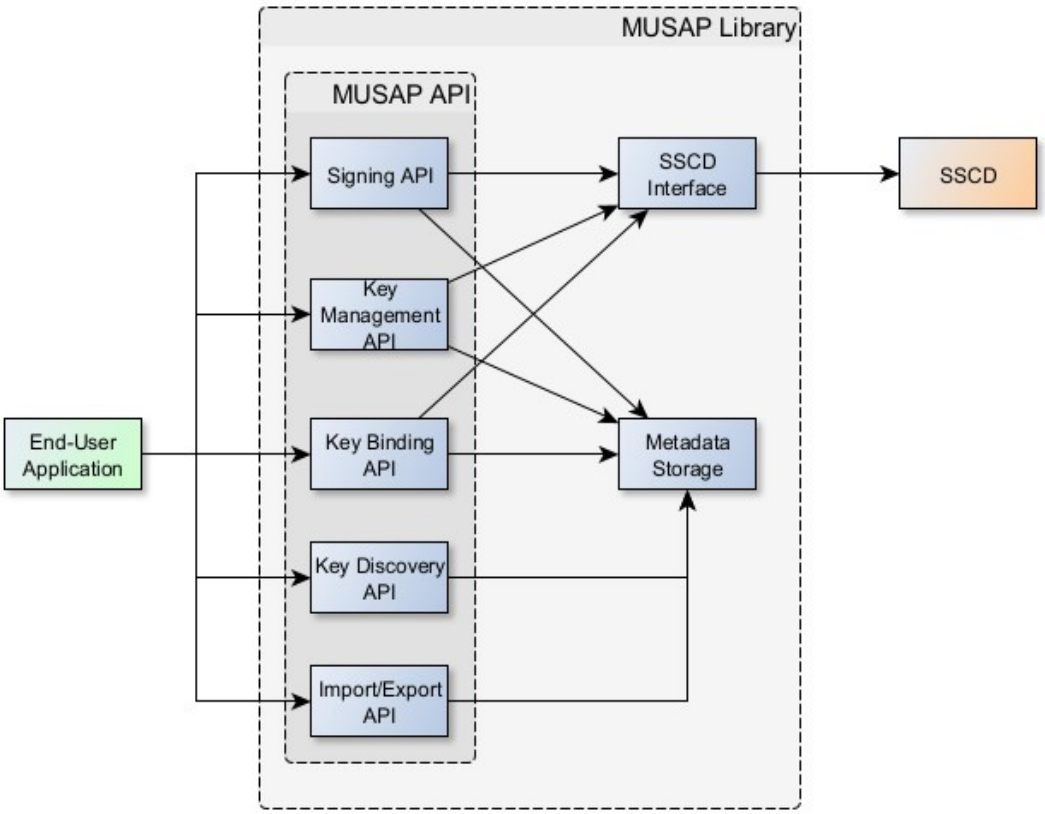

Figure 2: MUSAP API

3.2 Module 2: MUSAP Link

MUSAP Link is an optional back-end component that enables two way communication with the MUSAP Library. The primary goal of MUSAP Link is to provide signature APIs for web-based applications that do not have an app component.

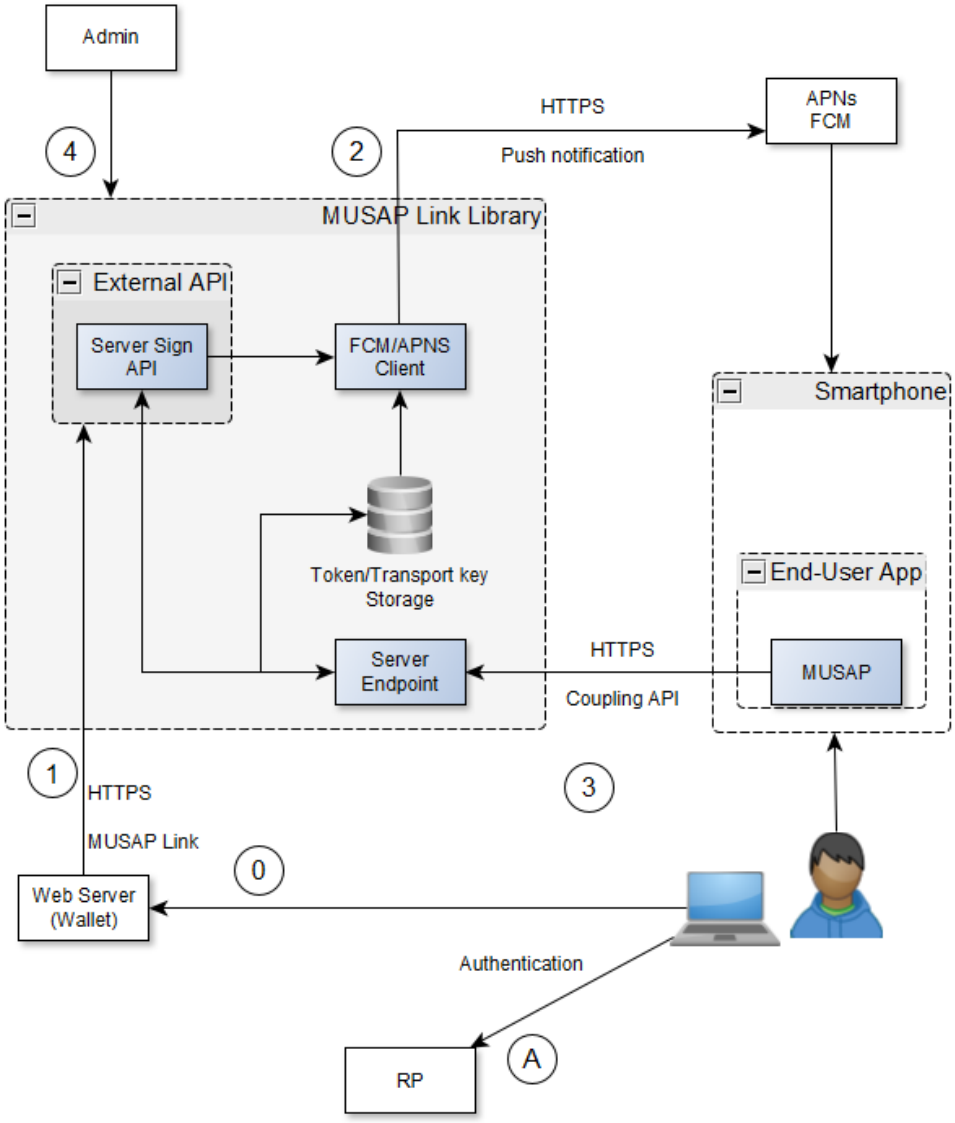

Figure 3: MUSAP Link architecture

4 KeyURI

KeyURI is an identifier that serves the following purposes:

1. to uniquely identify a key

2. to allow an application provider to request a specific key from an application

3. to list key properties

All keys in MUSAP have a Key URI. KeyURI specification [1] explains KeyURIs in detail.

5 Key Lifecycle

MUSAP provides several ways to manage key life-cycle.
The primary ones are:

1. Key Generation
2. Key Binding

MUSAP does not currently offer ways to revoke keys.

5.1 Key Generation

Key generation refers to cases where a new key-pair is generated to the user by the SSCD.

The key generation process is as follows:

1. The user starts key generation in End-User Application UI.
2. End-User Application has a criteria list of what SSCDs the user can use.
3. End-User Application prompts the user to choose which SSCD they want to use.
4. The user chooses one of the presented SSCDs, and provides any necessary data (e.g. keystore PIN).
5. End-User Application starts the key generation with MUSAP.
6. MUSAP requests the SSCD to generate a key with the given parameters.
7. SSCD generates the key.
8. MUSAP stores key metadata to the metadata storage. This could include public key, certificate, key alias etc.
9. MUSAP informs the End-User Application of key generation result.
10. End-User application informs user of the successful key generation.

5.2 Key Binding

Several SSCD devices perform their own independent key management operations like key generation. From MUSAP point of view these keys are pre-generated or externally managed keys. Examples of such SSCDs are Finnish Mobile ID, and eID cards. MUSAP needs a method to access those keys as well.

MUSAP employs Key Binding operations to enable MUSAP key access functionality for keys generated outside of the MUSAP API.

The key binding process is as follows:

1. The user starts key binding in End-User Application UI.
2. End-User Application has a criteria list of what SSCDs the user can use.
3. End-User Application prompts the user to choose which SSCD they want to use.

4. The user chooses one of the presented SSCDs, and provides any necessary data (eg. SSCD PIN).
5. End-User Application starts the key binding with MUSAP.
6. MUSAP calls the SSCD.
7. Depending on SSCD implementation the user may have to sign the existing key to prove they own it.
8. SSCD returns key metadata to MUSAP.
9. MUSAP stores key metadata to the metadata storage. This could include public key, certificate, key alias etc.
10. MUSAP informs the End-User Application of key binding result.
11. End-User application informs user of the successful key binding.

6 Signature

MUSAP signature process is as follows:

1. The user signing in End-User Application UI.
2. End-User Application has a criteria list of what SSCDs the user can use.
3. End-User Application prompts the user to choose which SSCD they want to use.
4. The user chooses one of the presented SSCDs, and provides any necessary data (eg. SSCD PIN).
5. End-User Application starts the key binding with MUSAP.
6. MUSAP calls the SSCD.
7. Depending on SSCD implementation the user may have to sign the existing key to prove they own it.
8. SSCD returns a signature to MUSAP
9. MUSAP returns the signature to the End-User Application.
10. End-User application informs user of the successful signature – and proceeds.

7 Key and SSCD Metadata

7.1 Storage Implementation

Current MUSAP apps store metadata to the phone storage. In the future, MUSAP can support alternative storage implementations such as a cloud storage.

7.2 Stored Metadata

MUSAP stores two types of metadata: SSCD metadata and key metadata. The metadata storage, such as a smartphone, can protect the metadata at rest by encrypting it with a symmetric key. This key can require user authentication such as biometrics.

7.2.1 SSCD Metadata

Parameter	Description	Example Value
SSCD Name	SSCD instance specific identifier, such as a serial number.	AAA-BBB-CCC
SSCD Type	Human readable name of the SSCD.	Mobile ID
SSCD ID	Type of the SSCD which uniquely identifies it.	SIM
Can generate key	Indicates if this SSCD can generate additional keys for the user.	true
Country	Which country the SSCD is based on. Stored in ISO 3166-1 alpha-2 format.	FI
Provider	Which TSP provides the SSCD.	Organization 1
Attributes	Arbitrary SSCD type specific attributes.	
Exportable	Are this SSCD and related public keys exportable?	true
Imported	Has this SSCD been imported previously?	false

7.2.2 Key Metadata

Parameter	Description	Example Value
Public Key	Public key as DER or JWK format.	"der": "...", “jwk”: “..:”
Certificate	Certificate of key.	"MII.."
Step-up policy	Policy that defines if, when, and how the user should	

	provide step-up authentication to use the key.	
Subject	Who is the subject of the key.	CN=John Doe
Key name	Human readable name of the key.	MUSAP Key 1
Key type	Type of	secp256k1
Created date	Date when the user created the key.	2023-08-23
Key URI	An identifier that See Key URI chapter.	mss:keystore=Mobiilivarmenne,storetype=ficom-mobileid,country=fi,loa=high,provider=DNA,keyname=MobileID-authn"
Certificate chain	Complete certificate chain of this key. This may be null if the key is self-issued.	[{ "subject": "CN=Root CA", "certificate": "MII.." }]
Key usages	Usages for which this key is intended.	digitalSignature
SSCD ID	SSCD instance specific identifier, such as a serial number.	AAA-BBB-CCC
SSCD Name	Human readable name of the SSCD.	Mobile ID
SSCD Type	Type of the SSCD which uniquely identifies it.	SIM
Attributes	Arbitrary SSCD type specific attributes.	
Roles	What are the roles defined for this key (RBAC)	personal
State	Key state. Currently defined states are ACTIVE, INACTIVE (key is permanently inactive), SUSPENDED (key cannot be used until restated), REVOKED (key certificate is revoked).	INACTIVE

7.3 SSCD Types

MUSAP supports at least the following SSCD types. MUSAP can support additional SSCD types via the SSCD Interface(s).

SSCD Type	Description
TEE	Local phone keystore. For example Android Keystore via Trusty or iOS Secure Enclave.
Dongle	NFC or USB dongle. For example Yubikey.
RemoteSign	HSM based remote signature. For example Alauda PBY.
UICC	UICC based signature. For example Alauda P38.
eUICC	eUICC based signature. For example Alauda P38 esim

Note: Depending on the device operating system, TEE is defined by Trusty or Secure Enclave for Android and iOS devices, respectively."

7.4 Import and Export {Optional}

Occasionally, user may want to switch old smartphone to new one. When MUSAP has been installed into the current phone, user needs a mechanism for moving the MUSAP and its SSCDs to a new phone.

MUSAP allows the user to export core parts of their data to be imported in another instance of MUSAP. This is an optional feature that the end-user application may support.

SSCDs that have been explicitly marked as unexportable will not be exported. An example of such SSCD is TEE (Android Keystore or iOS Secure enclave) which is bound to the original device.

To protect exported metadata, and to prevent fraudulent imports, the user can define a PIN during the export process. MUSAP uses the PIN to derive a symmetric encryption key and then encrypts the metadata with the key. During import, MUSAP asks the user for the PIN and runs the same key derivation to decrypt the metadata.

The SSCD metadata that can be exported includes:

SSCD Metadata		
Key	**Description**	**Example**
sscd_name	Human readable name of the SSCD	Yubikey
sscd_type	Type of the SSCD	NFC
sscd_id	Unique identifier of the SSCD instance	AAAA-BBBB-CCCC-DDDD
keygen_supported	Does this SSCD support new key generation?	true
country	Country of origin for this SSCD	FI
provider	Company that provides the SSCD service	Yubico
attributes	List of SSCD instance specific configuration parameters and other attributes	`{"name": "apikey", "value": "ffff"}`
keys	List of bound and generated keys associated with this SSCD	`{"key_name": "MUSAP Test", ... }`

Example

```
{
  "sscd_name": "Mobiilivarmenne",
  "sscd_type": "SIM",
  "sscd_id": "AAAA-AAAA-AAAA-AAAA",
  "keygen_supported": false
  "country": "FI",
  "provider": "DNA",
  "attributes": [
    {"name": "service-url", "value": "https://mobileid.example.fi/service"},
    {"name": "apikey", "value": "FFFFAAAA"},
```

```
    {"name": "ap-id",  "value": "musap-ap"}
  ],
  "keys": []
}
```

The key metadata that can be exported includes:

Key Metadata		
Key	**Description**	**Example**
key_name	Name of the key (given by the end-user)	MUSAP Test
key_algorithm	Key algorithm including key bits, type, etc.	RSA2048
created_dt	ISO-8601 timestamp when this key was created or bound	2023-10-24T00:00:000Z
updated_dt	ISO-8601 timestamp when this key was last updated.	2023-10-24T00:00:000Z
keyuri	KeyURI defining the key properties	`mss:keystore=Mobiilivarmenne,storetype=ficom-mobileid,country=fi,loa=high,provider=DNA,keyname=MobileID-authn`
publickey	Public key	`MII...`
subject	Certificate subject (optional)	CN=Test User
certificate	Certificate (base64)	`MII...`
cert_chain	Certificate chain including issuer and root	`[ { "subject": "CN=Root CA", "certificate": "MII..." } ]`
key_usages	X.509 KeyUsage(s)	`[ "digitalSignature" ]`
loas	Levels of Assurance supported by the key	`[ {"loa": "high", "scheme": "EIDAS-2014"} ]`
attestation	Key Attestation information	`{ "certificate": "MII..." }`
roles	Roles defined for the key	`[ "personal", "" ]`
state	Key state. Currently defined states are ACTIVE, INACTIVE (key is permanently inactive), SUSPENDED (key cannot be used until restated), REVOKED (key certificate is revoked).	`“inactive”`

7.5 Key Attestation {Optional}

MUSAP provides simlpe key attestation mechanisms that allows verifying that the signatures

originate from a SSCD. The mechanisms for this vary depending on the SSCD being used. MUSAP allows easy extension for new key attestation mechanisms for new SSCD integrations. Supported key attestation mechanisms include:

1. Attestation Certificate
2. Certificate Chain

8 MUSAP Link Server

MUSAP link is a server software that provides a way to communicate with a MUSAP app from a server. The most common use case for the link library is web-based wallets that need to provide signatures. See figure 3 below.

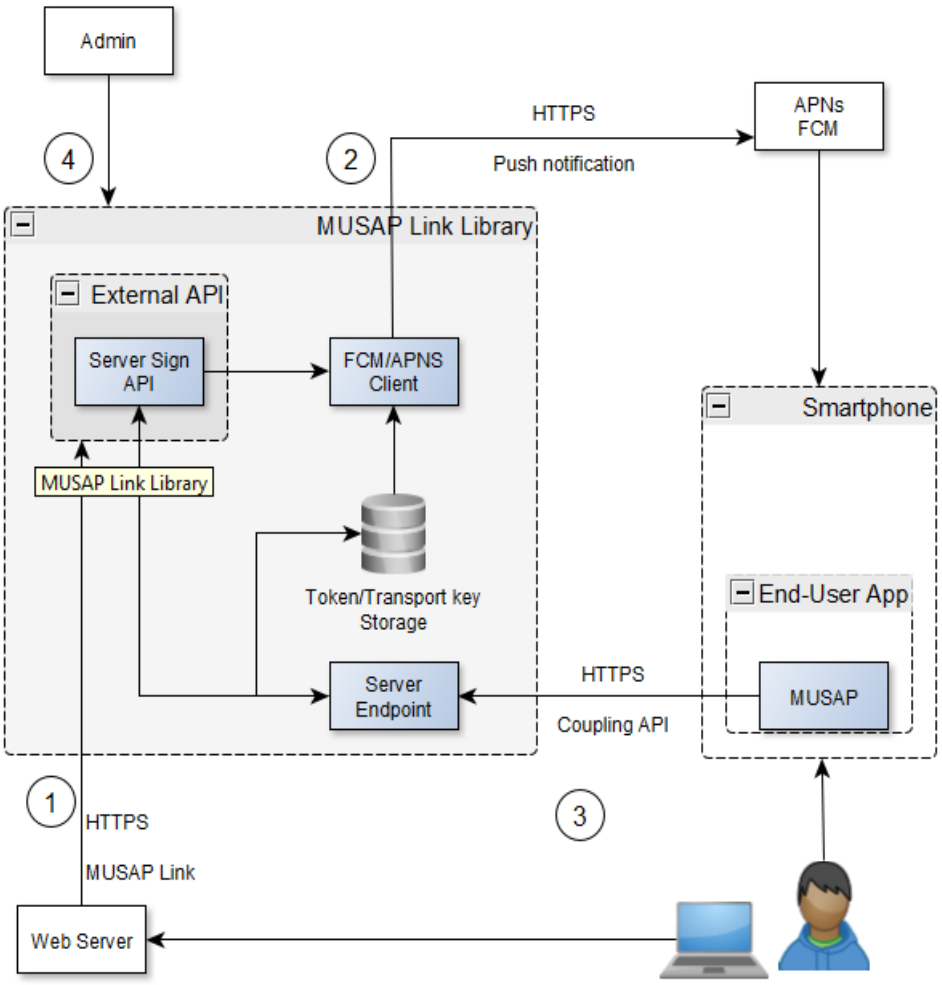

Figure 4: MUSAP Link overview

MUSAP link aims to be a simple gateway between the apps using MUSAP and web services.

MUSAP Link signature request flow is initialized by Web Wallet, which sends signature request to MUSAP Link server. MUSAP Link server sends a push notification to smartphone which awakes the End-User App and MUSAP API. Next the MUSAP API requests MUSAP Link for the signature request data. Connection between MUSAP API and MUSAP Link is called as MUSAP Coupling API. Finally, MUSAP sends the signature request data to SSCD. Signature response is send back to MUSAP Link and Web Wallet. See figure 4.

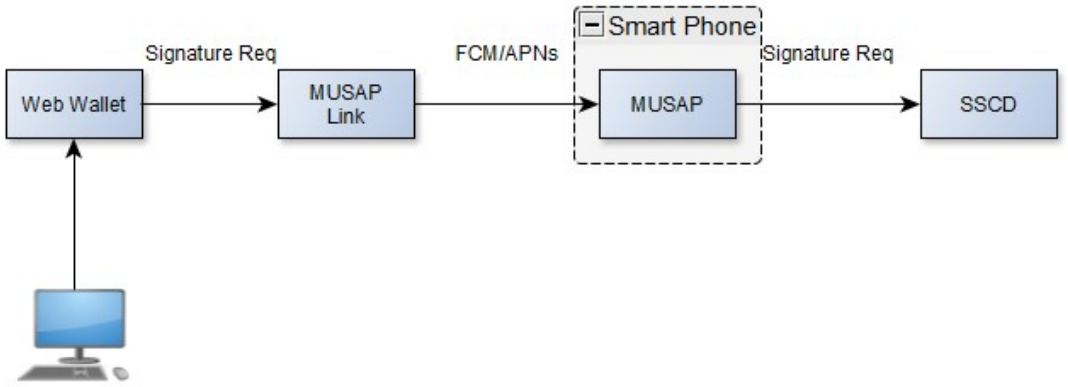

Figure 5: MUSAP Link basic signature request flow.

Each MUSAP link can be integrated with any number of End-User apps using the MUSAP library. The MUSAP link server URL (Coupling URL) must be configured on the MUSAP library to enable the coupling connection between the link library and the MUSAP library.

Figure 6: MUSAP Link registration over MUSAP Coupling API

If the MUSAP library has a configured link library, the MUSAP library will automatically register itself on the link library when new keys are generated. This registration process establishes transport encryption, delivers push notification tokens and generates a unique identifier for the MUSAP library. This unique identifier can be entered by the user on the web-wallet to allow the web-wallet to request signatures from the specific MUSAP library.

8.1 Transport Encryption

For additional security there is a transport security layer between MUSAP library and MUSAP Link. When enabled, all message payloads between them are encrypted with AES-CBC-PKCS7Padding cipher with a random IV. Messages also have a message authentication code to verify message integrity and authenticity. MUSAP library and MUSAP Link establish the encryption during enrolment.

The message payloads contain a nonce, and a timestamp. MUSAP Link rejects too old messages, and messages that have an already used nonce. These prevent replay attacks on MUSAP Link, and ensures that each payload is unique.

MAC uses encrypt-then-mac scheme. MUSAP library encrypts the payload, and then calculates a HMACSha256 of the payload, user identifier, message type, and AES encryption IV. When MUSAP Link, or MUSAP library receives a message, it first checks if MAC is correct by comparing the calculated and received MACs. They reject messages with a wrong MAC. The MAC proves the authenticity of the message.

9 MUSAP UI {Informative}

MUSAP includes UI elements that standardize user experience across different End-User Applications. For example, one MUSAP UI element asks user to enter their PIN for a SSCD for signing or key generation.

MUSAP UI should be easy to separate from the end-user app UI. Therefore, MUSAP UI has the following general principles:

1. Consistent terminology and unambiguous language.
2. Short sentences.
3. Localized texts and terms.
4. Texts comes to the fore (take center stage)
5. Single font, colour and size for everything.

10 MUSAP Link Interfaces

MUSAP Link has two interfaces: Link API for Relying Party and Coupling API for MUSAP on smartphone. See figure below.

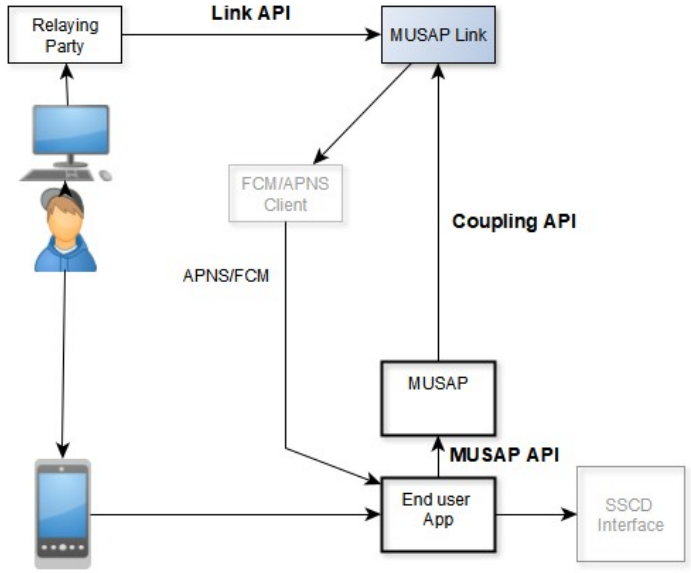

Figure 7: MUSAP Link interfaces.

10.1 Link API

This HTTP API is between the MUSAP link and a Relying Party (RP) that wants to request signatures (e.g. a web wallet).

HTTP method	URI	Arguments	Return Value	Description
POST	/link	None	(JSON) linkid, couplingcode, qrcode Example: { "linkid": "1467fd7e-2dff-4166-82bb-1ae98799ed74 ", "couplingcode": "AACCBB",	Start the linking process between a web application and an end-user application using MUSAP

			"qrcode": "" }	

POST	/sign	(JSON) data: Data to be signed as base64 encoded string	(JSON) keyid: identifier of the key that can be used in the future to request signatures with the key.
			publickey: Public key of the bound key.
			signature: signature made with the key to prove key ownership.
			Example:
			{
			"keyid": "FFFF"
			"publickey": {
			"pem": ""
			},
			"signature": {
			"raw": ""
			}
				Request a signature from MUSAP. The operation has to define which key to use and which MUSAP app instance to communicate with.

		(JSON) display: String that gets displayed to the user		

		(JSON) key: Object that helps MUSAP identify which key to sign with.
		Example:
		{
		"keyid": "FFFF",
		"keyname": "ABC",
		"algorithm": "RSA2K",
		"publickeyhash": "..."
		}		

		(JSON) linkid: Unique identifier telling which MUSAP app instance to communicate with. This is given to the service by the /link API. Example: "linkid": "A1234B"		
POST	/generatekey	(JSON) display: String that gets displayed to the		Request MUSAP to generate a

		user		new key. The operation can define required key features like algorithm.
		(JSON) key: Object that defines what kind of key to generate. Example: `{ "keyname": "ABC", "algorithm": "RSA2K", "keyusage": "testing" }`		
		(JSON) linkid: Unique identifier telling which MUSAP app instance to communicate with. This is given to the service by the /link API. Example: `"linkid": "A1234B"`		
POST	/docsign	(JSON) datachoice: Data to be signed as a list of objects that contain: - data (base64 encoded string) - key object identifying the key to sign		Request a signature from MUSAP. The operation can define multiple DTBS. MUSAP will sign the DTBS related to the key chosen by the user.
		(JSON) display: String that gets displayed to the user		
		(JSON) linkid: Unique identifier telling which MUSAP app instance to communicate with. This is given to the service by the /link API. Example: `"linkid": "A1234B"`		
POST	/updatekey	(JSON) linkid: Unique identifier telling which MUSAP app instance to		Request updating key details within MUSAP Link.

		communicate with. This is given to the service by the /link API. Example: `"linkid": "A1234B"`		Currently only used to update the key name.
		(JSON) keyid: KeyID of the key to update		
		(JSON) oldkeyname: Old Key Name of the key to update. Alternative to KeyID.		
		(JSON) newkeyname: New key name to set		

10.2 Coupling API

The MUSAP Coupling API is an HTTP API between the MUSAP library and the MUSAP link server. Coupling API is used when a server-side wallet tries to access a mobile SSCD on user's smartphone (see figure 2). This API is optional and only used when MUSAP link is enabled.

HTTP method	**URI**	**Arguments**	**Return Value**	**Description**
POST	/musap	(JSON) type, payload, musapid	(JSON) type, payload, musapid	Send a request to the MUSAP Link

The MUSAP Coupling API has one end-point that supports the following message types:

Type	**Request Payload**	**Response Payload**	**Description**
"getdata"	N/A	(JSON) type, payload, musapid	Check for pending requests in MUSAP Link.
"signcallback"	(JSON) linkid, publickey, signature	N/A	Response for the signature request
"signcallback"	(JSON) linkid, publickey, signature	N/A	Response for the key generation request
"enrolldata"	(JSON)	(JSON)	Enroll MUSAP

	fcmtoken, apnstoken, transid, tokendata	status	library to MUSAP Link
"updatedata"	(JSON) fcmtoken, apnstoken	(JSON) status	Update FCM/APNs token in MUSAP Link
"linkaccount"	(JSON) couplingcode, musapid	(JSON) status, linkid	Complete linking MUSAP to a RP service
"externalsignature"	(JSON) clientid, data, display, keyid, timeout, transid, attributes	(JSON) status, signature, publickey, attributes	Request a signature from an SSCD integrated with MUSAP Link

The MUSAP coupling API supports the following messages as a response to the GetData operation:

Type	**Request Payload**	**Description**

"sign"	(JSON)
	data,
	display,
	mode,
	format,
	linkid,
	attributes,
	key {
	  keyid,
	  keyname,
	  keyusage,
	  algorithm,
	  publickeyhash
	}
		Request a signature from MUSAP

11 MUSAP API Description

MUSAP library has the following APIs for the End-User Application.

11.1 General API

General API has functions that interact directly with MUSAP, and not with any external modules.

Function	Arguments	Return Value	Description
init(arg1)	(Context) arg1: End-User App Context	void	Initialize MUSAP library for use. End-User Application must call this upon startup.
enableSscd(arg1)	(MusapSscdInterface) arg1: SSCD to enable on End-User application.	void	Enable a SSCD for End-User Application. Enabled means MUSAP can do either key binding or key generation on the SSCD.
setDebugLog(arg1)	(boolean) arg1: true enables debug logging, false disables it.	void	Enable or disable debug logging of MUSAP.

11.2 Key Discovery API

Key Discovery API has functions that relate to SSCD and key discovery.

Function	Arguments	Return Value	Description
listKeys	No arguments	List<MusapKey>	List all generated or bound keys on this device.
listKeys(arg1)	(Key Search Request) arg1: Request with criteria for listed keys.	List<MusapKey>	List all generated or bound keys that match request criteria on this device.
listActiveSSCDs	No arguments	List<MusapSscd>	Lists all active SSCDs on this device. Active means MUSAP has done either key binding or key generation on the SSCD.
listActiveSscds(arg1)	(SSCD Search Request) arg1:	List<MusapSscd>	Lists all active SSCDs that match the

	Request with criteria for listed SSCDs.		request criteria on this device. Active means MUSAP has done either key binding or key generation on the SSCD.
listEnabledSscds	No arguments	List<MusapSscd>	Lists all enabled SSCDs on this device. Enabled means MUSAP can do either key binding or key generation on the SSCD.
listEnabledSscds(arg1)	(SscdSearchReq) arg1: Request with criteria for listed SSCDs.	List<MusapSscd>	Lists all enabled SSCDs that match the request criteria. on this device. Enabled means MUSAP can do either key binding or key generation on the SSCD.
getKeyByUri(arg1)	(String) arg1: KeyURI of the wanted key.	MusapKey	Get a single generated or bound key by its KeyURI.

11.2.1 Key Search Request

The key search request is defined as follows:

KeySearchReq			
Parameter	**Type**	**Description**	**Example Value**
SSCD Type	String	See SSCD Types	Local Keystore
SSCD Provider	String	Who provides the SSCD. For example "AKS" for Android Keystore.	AKS
SSCD Country	String	ISO 3166-1 country code (alpha-2 format) of the SSCD	FI
Key Algorithm	Object	Desired key algorithm. See Key Algorithms.	KeyAlgorithm.ECC_P256_K1
Signature Algorithm	Object	Signature algorithm that the key should support. See Signature Algorithms.	SignatureAlgorithm.SHA256_WITH_ECDSA
LoA	Object	Desired LoA of the key.	MusapLoA.EIDAS_SUBSTANTIAL
Alias	String	Exact key alias	TestKey

DID	String	Which DID the key has been generated for.	did:method:value
KeyURI	String	Partial or full KeyURI	mss:alias=TestKey
SSCD Attribute	String, String	Arbitrary attribute of the SSCD. This is SSCD specific.	managementkey, FFFFFFFF
Key Attribute	String, String	Arbitrary attribute of the Key. This is key specific.	keyslot, 1
Role	String	Role the keys have been generated for	personal

11.2.2 SSCD Search Request

The SSCD search request can be used to search for an activated SSCD.

KeySearchReq			
Parameter	**Type**	**Description**	**Example Value**
SSCD Type	String	See SSCD Types	Local Keystore
SSCD Provider	String	Who provides the SSCD. For example "AKS" for Android Keystore.	AKS
SSCD Country	String	ISO 3166-1 country code (alpha-2 format) of the SSCD	FI
Key Algorithm	Object	Desired key algorithm. See Key Algorithms.	KeyAlgorithm.ECC_P256_K1
Signature Algorithm	Object	Signature algorithm that the key should support. See Signature Algorithms.	SignatureAlgorithm.SHA256_WITH_ECDSA
LoA	Object	Desired LoA of the key.	MusapLoA.EIDAS_SUBSTANTIAL

11.3 Signature API

Signature API contains the function to produce signatures with any MUSAP SSCD.

Function	**Arguments**	**Return Value**	**Description**
sign(arg1, arg2)	(SignatureReq) arg1: Signature request that includes Data-to-be-Signed, signing, algorithm, SSCD LoA etc.	void	Requests MUSAP to produce a signature with a certain SSCD. This function is asynchronous. MUSAP will notify the callback parameter when complete.
	(MusapCallback<MusapSignature>) arg2: Callback that MUSAP		

	calls when signature is either completed or failed.		

11.3.1 Signature Request

The signature request contains mainly the key, data and signature algorithm to use.

SignatureReq			
Parameter	**Type**	**Description**	**Example Value**
Key	Object	MUSAP key to sign with	N/A
Data	byte[] or list of byte[]	Data to sign as a byte array. May contain a list of data to sign. Each list element will be signed.	FFFF
Signature Algorithm	Object	Signature algorithm to use. Make sure that this matches the supported algorithms of the key.	SignatureAlgorithm.SHA256_WITH_ECDSA
Signature Format	Object	Format of the signature. If not specified, default is RAW.	SignatureFormat.CMS

11.4 Key Lifecycle API

Key Lifecycle API has functions that relate to key generation and key removal.

Function	**Arguments**	**Return Value**	**Description**
generateKey(arg1, arg2, arg3)	(MusapSscdInterface) arg1: Which SSCD should generate the key	void	Request MUSAP to generate a key on a certain SSCD. This function is asynchronous. MUSAP will notify the callback parameter when complete.
	(Key Generation Request) arg2: Key generation request that contains algorithm, key alias, SSCD LoA, etc.		
	(MusapCallback<MusapKey>) arg3: Callback that MUSAP calls when key generation is either completed or failed.		

removeKey(arg1)	(MusapKey) arg1: Previously generated MUSAP key to remove.	boolean	Remove a previously generated key from MUSAP. Return value indicates a successful removal.
updateKey(arg1)	(UpdateKeyReq) arg1:	boolean	Update a bound or generated key metadata. Return value indicates a successful update.
removeSscd(arg1)	(MusapSscd) arg1: SSCD to remove.	boolean	Remove a previously enabled SSCD. Return value indicates a successful removal.

11.4.1 Key Generation Request

The key generation request is used when requesting key generation.

KenGenerationReq			
Parameter	**Type**	**Description**	**Example Value**
Key Algorithm	Object	Desired key algorithm. See Key Algorithms. This includes key size.	KeyAlgorithm.ECC_P256_K1
Alias	String	Exact key alias	TestKey
DID	String	Specify the DID to generate the key for.	did:method:value
Key Attribute	String, String	Arbitrary attribute of the Key. This is key specific.	keyslot, 1
Role	String	Role to generate the key for	Personal
Step-up policy	Object	Step-up policy that states when/if step-up authentication is used when accessing the key.	{ "frequency": "ALWAYS", "validity": "10 min", "type": "biometric" }

11.4.2 Key Update Request

Key update request can update the metadata of an already existing key.

KeyUpdateReq			
Parameter	**Type**	**Description**	**Example Value**
Alias	String	Exact key alias	TestKey
Key	MusapKey	Which key to update.	

DID	String	Specify the DID to generate the key for.	did:method:value
Key Attribute	String, String	Arbitrary attribute of the Key. This is key specific.	keyslot, 1
Role	String	Role to generate the key for	Personal
State	String	Update key state to one of: ACTIVE, INACTIVE (key is permanently inactive), SUSPENDED (key cannot be used until restated), REVOKED (key certificate is revoked).	INACTIVE

11.5 Key Binding API

Key Binding API allows MUSAP to use an already existing key.

Function	Arguments	Return Value	Description
bindKey(arg1, arg2, arg3)	(MusapSscdInterface) arg1: Which SSCD should bind the key	void	Request MUSAP to bind a key on a certain SSCD. This function is asynchronous. MUSAP will notify the callback parameter when complete.
	(Key Binding Request) arg2: Key binding request that contains algorithm, key alias, SSCD LoA, etc.		
	(MusapCallback<MusapKey>) arg3: Callback that MUSAP calls when binding is complete or failed.		

11.5.1 Key Binding Request

The key binding request is used when binding MUSAP to an existing key on an SSCD.

KeyBindReq			
Parameter	**Type**	**Description**	**Example Value**
Alias	String	Exact key alias	TestKey
DID	String	Specify the DID to generate the key for.	did:method:value

Key Attribute	String, String	Arbitrary attribute of the Key. This is key specific.	keyslot, 1
Role	String	Role to generate the key for	Personal
Step-up policy	Object	Step-up policy that states when/if step-up authentication is used when accessing the key.	{ "frequency": "ALWAYS", "validity": "10 min", "type": "biometric" }

11.6 Import/Export API

Import/export API allows the End-User Application to transfer metadata to another End-User Application.

Function	Arguments	Return Value	Description
importData(arg1)	(String) arg1: MUSAP data from another device. Return value of exportData.	void	Import data from another MUSAP app to this End-User Application. Example use case is user changing their device.
exportData	No arguments	String	Export key and SSCD metadata from this End-User Application for use in another app. Example use case is user changing their device.

11.7 SSCD Interface(s)

The SSCD interface allows End-User Application to add any SSCD for use with MUSAP. End-User Application should not call these methods directly, but instead use other MUSAP APIs which can call the new SSCD.

Function	Arguments	Return Value	Description
bindKey(arg1)	(KeyBindReq) arg1: Key generation request that contains algorithm, key alias, SSCD LoA, etc.	MusapKey	Bind a key on SSCD with MUSAP.
generateKey(arg1)	(KeyGenReq) arg1: Key generation request that contains algorithm, key alias, SSCD LoA, etc.	MusapKey	Generate a new key on the SSCD.
sign(arg1)	(SignatureReq) arg1:	MusapSignature	Produce a new

	Signature request that includes Data-to-be-Signed, signing, algorithm, SSCD LoA etc.		signature on the SSCD.
getSscdInfo	No arguments	MusapSscd	Get info of this SSCD such as supported signing algorithms, SSCD type and name, etc.
isKeyGenSupported	No arguments	boolean	Returns true if the SSCD can generate new keys and false otherwise.
getSettings	No arguments	SscdSettings	Return a map of settings this SSCD can use.

11.8 Coupling API

This call API is provided by the MUSAP library to the Android/iOS app.

Function	Arguments	Return Value	Description
enableLink(arg1)	arg1: URL that points to the link library	MusapLink object that contains link properties	Enable the MUSAP link connection. Sets the URL to MUSAP link and an ID to use during linking.
getLink()	No arguments	MusapLink object or null	Get the details of the enabled MUSAP Link.
getLinkId()	No arguments	MUSAP Link ID	Get a Link ID used to bind a web application
disableLink()	No arguments	void	Disable the MUSAP link
isLinkEnabled()	No arguments	Boolean	Check if MUSAP link is enabled
pollLink()	No arguments	Signature Request or null	Poll the MUSAP link for a signature request. This should be called periodically after receiving a notification.
coupleWithRelyingPartY(arg1, arg2)	(String) arg1: Coupling code that the user acquired from a	void	Couple this instance of a MUSAP app with a relying party.

	relying party		
	(MusapCallback<RelyingParty>) arg2: Callback that MUSAP calls when coupling is complete or failed		
updateFcmToken(arg1, arg2)	(String) arg1: New Firebase or APNS token of this app instance.	void	Update the MUSAP Link database with this apps latest identifier used for push notifications.
	(MusapCallback<MusapLink>) arg2: Callback that MUSAP calls when update is complete or failed.		
listRelyingParties()	No arguments	List<RelyingParty>	List all Relying Parties this MUSAP app has been coupled with.
removeRelyingParty(arg1)	(RelyingParty) arg1: Previously coupled relying party	boolean	Remove a previously coupled Relying Party. Return value indicates success.
sendSignatureCallback(arg1, arg2)	(MusapSignature) arg1: Signature produced by a user SSCD.	void	Send a successful signature to MUSAP Link so it knows that the signature is successful.
	(String) arg2: Transaction ID of the signature request		
sendKeygenCallback(arg1, arg2)	(MusapKey) arg1: Key metadata generated by a user SSCD.		Send a successful key generation response to MUSAP Link so it knows that the key generation is successful.
	(String) arg2: Transaction ID of the key generation request		

12 Error Handling

If any MUSAP function fails, it throws a MusapException which contains an error code.

The same error codes are available in all MUSAP interfaces (LINK API, Coupling API and MUSAP API).

These codes should be localized when displayed to the user. No other localized error codes exists.

Below table lists all possible error codes and their descriptions.

Error Code	Error Name	Description
101	wrong_param	Request contains an invalid parameter. Error message will explain more.
102	missing_param	Request is missing a required parameter. Error message will explain more.
105	unknown_key	User's MUSAP reports that the key is not found.
107	unsupported_data	The SSCD does not supported given data or signature algorithm/hash algorithm combination.
108	keygen_unsupported	The selected SSCD does not support key generation.
109	unknown_user	Requested user is unknown.
208	timed_out	The request timed out.
401	user_cancel	User cancelled the request.
402	key_blocked	User's key is in a blocked state. For example, PIN blocked.
403	sscd_blocked	User's SSCD is in a blocked.
404	sscd_unreachable	User's SSCD is unreachable.
405	coupling_error	Coupling error occurred.
900	internal_error	Internal MUSAP error. Try again later.

13 References

[1] KeyURI specification