Procilon

Server Signing

Integration Guide

CP5

utimaco®

Imprint

Copyright 2025	Utimaco IS GmbH
	Germanusstr. 4
	D-52080 Aachen
	Germany
Phone	AMERICAS +1-844-UTIMACO (+1 844-884-6226)
	EMEA +49 800-627-3081
	APAC +81 800-919-1301
Internet	https://support.hsm.utimaco.com/
e-mail	support@utimaco.com
Document Version	1.0.0
Date	2025-07-23
Status	PUBLISHED
Document No.	IG-2025-0037

Table of Contents

1 Scope

This document describes tasks and procedures for the integration of server signatures with the product solution of the procilon Group.

In the following, the term server signature is used synonymously and in the sense of the term remote signatures.

2 General Requirements

Server signatures are specified in the protection profile (PP) DIN EN 419241-2:2019. The PP describes the basic processes, the requirements for an identity provider, the components involved, and the possible scope for decision making during implementation. The following figure shows the basic structure of a server signature solution based on this:

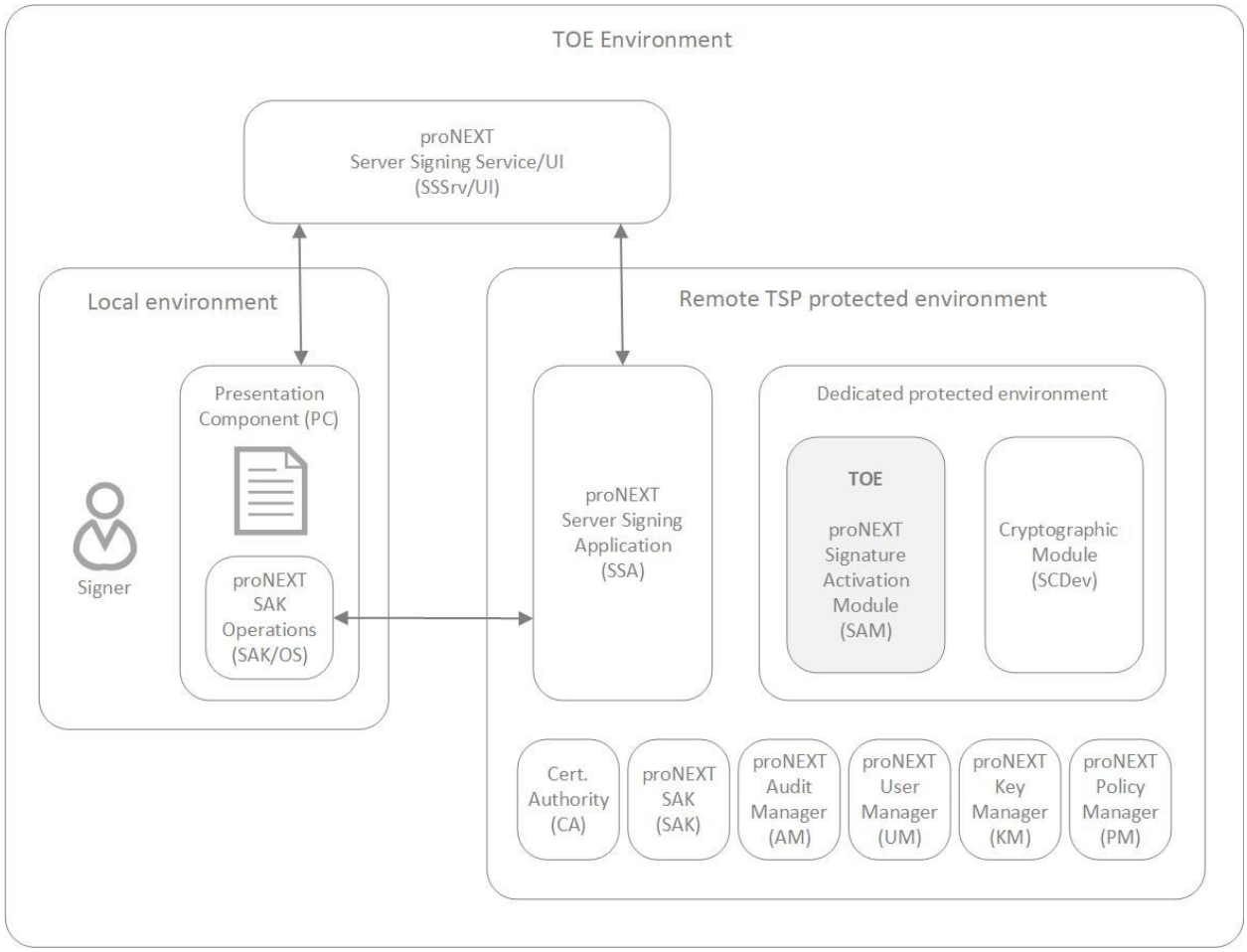


Figure 1: Server signing that meets the SCAL2 requirements

A system as the one shown in the previous figure is called 'trustworthy system supporting server signing' (TW4S) in the context of server signatures and the protection profile (PP) DIN EN 419241-2:2019. This provides remote digital signatures as a service and ensures that the signer's signature keys are only used for their intended purpose under the sole control of the signer in question.

The TW4S uses a cryptographic module to generate the signature key and produce the digital signature value. The system basically consists of a local and a remote environment. The signer resides in the local environment and interacts with the Server Signing Application (SSA) in the

remote environment. The Signer Interaction Component (SIC) used for this purpose can be provided conceptually and depending on both the signer's side and on the remote side of the TW4S.

The purpose of the interaction between the signer and the SSA is to enforce the use of a specific signature service addressed by the SSA. The signature process is performed using what is known as the Signature Activation Protocol (SAP), which requires the provision of Signature Activation Data (SAD) using the SIC.

The SAD combines three elements that are essential for the provision of a remote signature: the signer authentication (performed according to [EN419241-1] SCAL.2), the signature key chosen by the signer for the signature process and the data to be signed (DTBS/R(s)).

To ensure that the signer has sole control over his signature keys, the signature operation must be authorized. This task is performed by a Signature Activation Module (SAM) that can process an endpoint from SAP, verify SAD, and activate the signature key within a cryptographic module. Both the cryptographic module and the SAM must be located within a dedicated protected environment. SAD verification here means that the SAM verifies the binding between the three SAD elements as well as the authentication of the signer. This binding of the SAD components is ensured, among other things, by the signer using a temporary key created in its environment, which is introduced into the signer authentication data in the form of an ID token on the one hand and is used to sign the SAD on the other.

The signer is authenticated by means of indirect authentication by the SAM. In this process, an authentication service in the form of an identity provider verifies the signer's authentication factors and then issues an assertion stating that the signer has been successfully authenticated. The SAM verifies the assertion issued by the identity provider or the Identity Token contained therein. In doing so, it must be able to rely on the assertion or the service issuing it.

The signer may use a user interface to display documents. The Signer Interaction Component (SIC) is accessed by the signer through the user interface. The SIC, in turn, is used to communicate with the Server Signing Application (SSA). The SSA forwards communication from the SIC to the QSCD. Within the QSCD, the SAM receives the messages. Once the SAM module has verified the SAD, it can authorize the activation of the signature key within the cryptographic module and generate a digital signature value. The value is given via the SSA to the SIC, which packs the value into a signature container and returns it to the signer via the user interface.

The SAM also generates audit logs. In addition, a TW4S relies on other services:

- signers are identified and registered.
- Signature keys are certified by a certification authority.

- The signature creation application is responsible for creating the signed document using the signature values provided by the TW4S.

Together with the CC certified and eIDAS compliant Utimaco HSM of the model family 'CryptoServer Se-Series Gen2 CP5' (CC certificate [CCertCP5]), the proNEXT SignatureActivationModule v1.0.0 (CC certificate [CCertSAM]), which is also CC certified, forms the QSCD to be used in the context of remote signature.

3 Usage Scenarios

The usage scenarios defined in [EN419241-2] whose functionality is realized by the server signing are:

(US1) Privileged User Creation

(US2) Signer Creation

(US3) Signer Maintenance

(US4) Key Pair Generation

(US5) Key Pair Deletion

(US6) Signing

(US7) SAM Maintenance

These usage scenarios form the basis for the following descriptions of the server signing processes and their integration.

3.1 Privileged User Creation

Privileged User Creation is the usage scenario in which a privileged user creates a new privileged user.

The privileged user firstly initiates a request for user authentication at the SSSrv/UI where the user authentication is performed using the IdP. The request for Privileged User Creation is then send by the Privileged User to the SSSrv/UI and then goes its way along to the SAM. When retrieving the request, the SAM service checks if the requestor is authenticated and identified correctly. If the authenticity and identity check is successful, the SAM service creates a new Privileged User based on the data it received with the request.

Therefor, the SAM stores the given user information in the form of a user entry in the SAM DB.

3.2 Signer Creation

Signer Creation is the usage scenario in which a user registers with TW4S and is created as a signer.

As a starting point for the usage scenario, the functionality of the SAM allows a user to register to use the TW4S to become signer. In this context, the SSSrv/UI provides the functionality to

perform the identification of a user or to have this performed with recourse to appropriate services for mapping both eID based and hardtoken based authentication. Provided that the identification of the user has been successfully performed, the SAM initiates the creation of the signer.

3.3 Signer Maintenance

Signer Maintenance is the usage scenario in which a privileged user or signer updates the attributes of a signer (e.g. his own).

The starting point of this scenario is that a privileged user or signer initiates a request for user authentication at the SSSrv/UI where the user authentication is performed using via the IdP. Optional when the user is a privileged user the signer is selected whose attributes should be updated. The request for updating attributes of a signer then is send by the privileged user or signer to the SSSrv/UI and then goes its way along to the SAM. When the request is retrieved, the SAM verifies that the requestor is authenticated and identified. If the authenticity and identity check is successful, the SAM maintains a Signer's user record based on the data it received with the request. The SAM updates a Signer's authentication data in the form of assigned certificates in the SAM DB.

3.4 Key Pair Generation

Key Pair Generation is the usage scenarios in which a privileged user or signer generates a pair of signing keys and assigning them to a signer (e.g. his own).

The starting point of this scenario is that a privileged user or signer initiates a request for user authentication at the SSSrv/UI where the user authentication is performed via the IdP. Optional when the user is a privileged user the signer is selected whose attributes should be updated. The request for generating a new key pair for a signer then is send by the privileged user or signer to the SSSrv/UI and then goes its way along to the SAM. When the request is retrieved, the SAM verifies that the requestor is authenticated and identified. If the authenticity and identity check is successful, the SAM generates the Signer's key pair. The SAM requests the cryptographic module to generate a key pair. The cryptographic module generates the Wrapped Key based on the generated key pair, signs the Wrapped Key, generates a certificate request based on the generated key pair, signs the certificate request and sends both the Wrapped Key and the certificate request back to the SAM which then requests the storage of the Wrapped Key in the Key Manager, stores the signed certificate request in the KM. The user a privileged user or signer then gets the result of its request by confirmation.

3.5 Key Pair Deletion

Key Pair Deletion is the usage scenarion in which a privileged user or signer deletes the signing key and related key information (in particular the public key) as the relation the a signer (e.g his own).

The starting point of this scenario is that a privileged user or signer initiates a request for user authentication at the SSSrv/UI where the user authentication is performed via the IdP. Optional when the user is a privileged user the signer is selected whose attributes should be updated. The request for deleting a key pair of a signer then is send by the privileged user or signer to the SSSrv/UI and then goes its way along to the SAM. When the request is retrieved, the SAM verifies that the requestor is authenticated and identified. If the authenticity and identity check is successful, the SAM deletes the Signer key pair referenced by the data it received through the request. The SAM requests the deletion of the Wrapped Key associated with the Signer key pair to be deleted from the KM. Since Wrapped Keys do not contain keys in plaintext with that all is done.

3.6 Signing

Signing is the usage scenario in which the signer initiates a request for remote signature of data and, using and activating a suitable signature key, has the remote signature executed, with the result that the signer then receives the remote signature data back as a response.

The starting point of this scenario is that the signer initiates a request to perform a remote signature from the business application he or she is using. The Signer Interaction Component (SIC) for the communication between signer and SSA as remote peer to be addressed in the remote environment can be provided conceptually and depending on the usage scenario both on the side of the signer and on the remote side of the TW4S. The request for remote signing is sent to the Signature Activation Module (SAM) via the Server Signing Application (SSA). This verifies the request and the requester. If the verification is successful, the signature key assigned to the signature process via Signature Activation Data (SAD) is activated and the remote signature is triggered. The request information must be created in the local environment before the request is sent to the SAM via the SSA. Before the signature process is triggered, the signer must also confirm his intention to perform the remote signature. Authentication of the signer requires that the signer has previously identified himself to an identity provider and that the identity provider has issued him an identity token.

To ensure that the signer has sole control over his signature keys, the signature process must be authorized. This task is performed by the Signature Activation Module (SAM), which can verify SAD and activate the signature key within a cryptographic module. SAD verification here means

that the SAM verifies the binding between the three SAD elements as well as the authentication of the signer. This binding of the SAD components is ensured, among other things, by the signer using a temporary key created in its environment, which is introduced into the data for signer authentication in the form of an ID token on the one hand and is used to sign the SAD on the other.

The flow of the remote signing process is basically as follows:

- Authentication of the signer
  - Authentication of the signer to an identity provider Issuance of ID token for the signer by the identity provider
- Selection of signature information
  - Requesting a list of key IDs suitable for remote signing together with the associated certificate
  - Selection of the key ID/assigned certificate to be used for the upcoming remote signature
- SAD creation
  - Creation of the temporary key in the signer's environment
  - Extension of the ID token with the public part of the temporary key and the remote signature scope
  - Verification of the document to be signed
  - Initiation of remote signature using extended ID token
  - Generation and signature of the SAD
  - Transfer of SAD or the SAD components bound by SAD signature to the remote signature process
- Activation of the signature key
  - Verification of signer authentication
  - Checking the validity of the declaration of intent for remote signature
  - Request of the wrapped key matching the key ID
  - Verification that key ID and user ID match (SAD vs. wrapped key vs. ID token)
  - Activation of the signature key
- Creation of the signature value

- Creation of the signature value by the cryptographic module
- Verification of the signature value and the signer certificate
- Creation of the signature container for embedding the signature value
- Return of the document signature to the signer

The actual process and the actions that take place are explained in detail in the chapter about the signing process, including a sequence diagram and step-by-step description.

3.7 SAM Maintenance

The SAM Maintenaince usage scenario includes administrative functions for basic management of the Signature Activation Module (SAM).

The administrative functions include:

- Check Code/Data Integrity: checking code/data integrity of all modules of the SAM subsystem before startup
- Start / Stop: starting and stopping the the SAM (service)
- Initialize SAM Firmware / Set IdP Public Keys: Initializing of SAM Firmware and importing of IdP Public Keys to it

The administrative functions are implemented by management scripts which can be called by two authorized administrators via the SAM Management Command Line Interface.

The operation Check Code Integrity is carried out for the subsystem of the SAM Service module and for the SAM MAN module by the Linux command sha512sum. The expected checksums are stored in a configuration file. The expected checksums are stored in a configuration file, which must be created in advance using the shell script checksumSAM.sh.

The operation Check Data Integrity is performed by calling the appropriate methods via the REST Service provided by the module SAM Service.

The operation Set IdP Public Keys imports a certain number of public keys to the SAM firmware (when it is started) to verify the ID tokens issued and signed by the IdP. 'Initialize SAM Firmware' initializes the SAM Firmware and for example derives key material from the used MBK necessary for providing the functions of the SAM Firmware.

4 Server Signing Architecture

4.1 Overview

The environment presented in chapter 1 is the basis for the server signing architecture.

In addition, [EN419241-2] specifies the requirements for possible users, components and operations provided by server signing using a so called Signature Activation Module (SAM).

The resulting architecture with the environment, requirements and components provided by procilon is shown below:

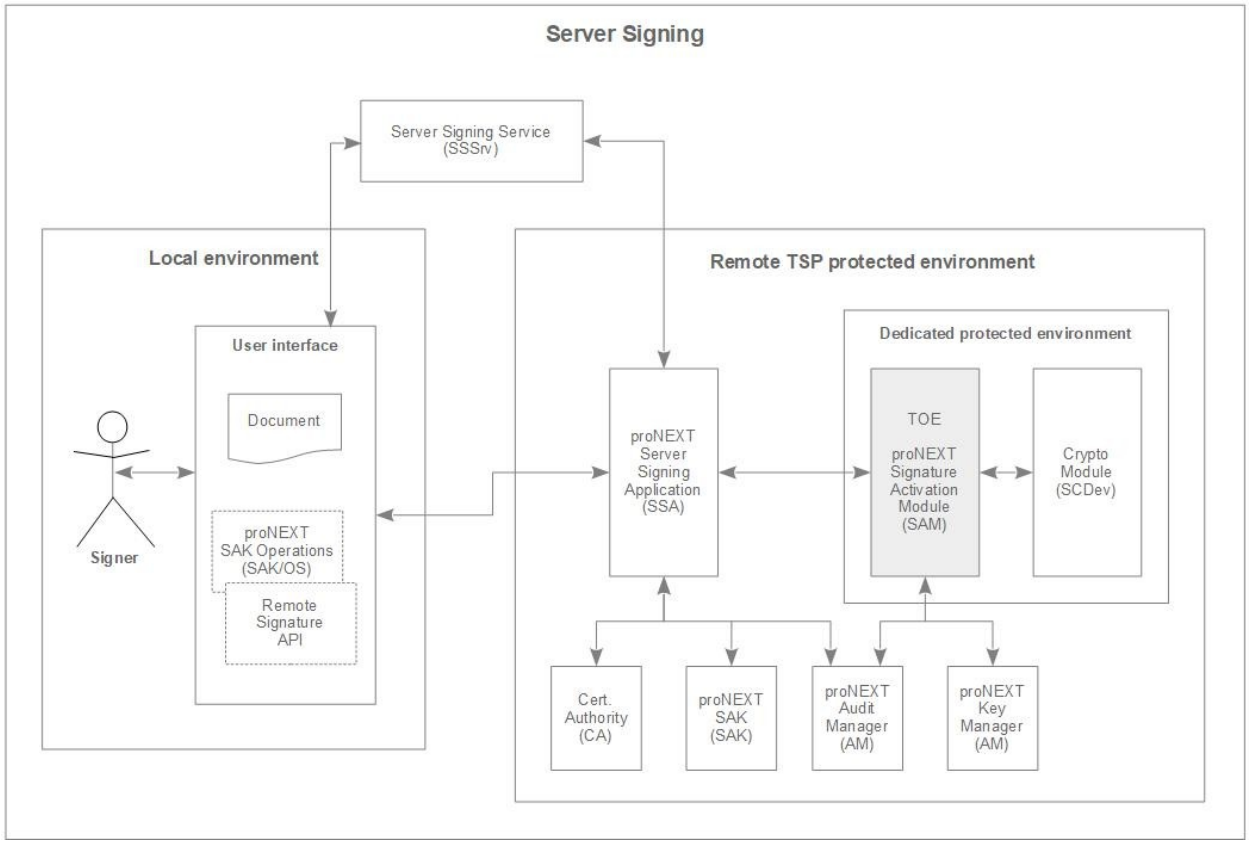


Figure 2: Server Signing Architecture

In this architecture, the signer resides in the local environment and uses a user interface provided by a module called Server Signing Service (SSSrv). In the context of server signatures, the SSSrv acts as a signature creation application. The user interface displays the document to be signed and other relevant data for the signer. This includes, among other things, the document hash, the signature key selected for signature creation, and the associated signer certificate.

The component SAK operations (SAK/OS) as well as the Remote Signature API (RSAPI), which in the context of server signatures correspond to the Signer Interaction Component (SIC) according to [EN419241-1], can be used to generate the signature activation data (SAD) and to communicate (e.g., send requests by the signer) with the SSA. The SSA interacts directly with the SAM and forwards communications from the SAK/OS to it. It requires successful identification and authentication of the signer before allowing any actions that may affect the SAM or signature key.

Within the dedicated protected environment, the SAM receives requests forwarded by the SSA and processes requests regarding verification of these. When the SAM successfully verifies the SAD provided by a Signer, it can authorize the activation of the signature key associated with the signature operation within the cryptographic module and have a digital signature value generated by the cryptographic module. The value is returned to the SSA and further delivered to the SAK/OS after verification.

An attached CA must generally comply with the requirements of [EN319401], [EN319411-1] and [EN319411-2].

4.2 Components

The components of the server signing architecture already mentioned are described in more detail below:

Abbreviation	Name	Description
AM	proNEXT Audit Manager	Is a service for providing audit management functions for the components of the server signing architecture. The functions of the service include the delivery of audit entries via the REST interface, verification and export of the audit log, ensuring integrity protection in s and configurations. The Audit Manager is used in particular by the Server Signing Application (SSA) and the Signature Activation Module (SAM).

CA	Certification Authority	Provides certificate services in the sense of a certification authority. In particular, it provides the additional services required for a TW4S, such as: ▪ the identification and registration of signers and ▪ the certification of signature keys
SCDev	Cryptographic Module	Used to create both signer signature keys and signatures (signature values) requested by the signer. Located in a specially secured environment within the TW4S remote environment. In the context of remote signing, it is mandatory to use an HSM of the model family 'CryptoServer Se-Series Gen2 CP5' (CC certificate [CCertCP5]) in conjunction with the also CC certified proNEXT SignatureActivationModule v1.0.0 (CC certificate [CCertSAM]) as a so called QSCD.
KM	proNEXT Key Manager	Provides functions that enable the creation, management and retrieval of key material. Objects managed by KM, so called managed objects, consist of a unique ID and a binary, which can represent certificates, public keys and user objects, among other things. Links between the managed objects are used to link them and can, for example, represent relationships such as the ownership.
RS API	Remote Signature API	Software that is installed in the signer's environment. Can be an application accessed from a browser or a mobile device, for example. Participates in the signature activation protocol (SAP) and generates the SAD. Provides the link between the signer and the signature process (linking the document to be signed, the remote signature key used by the signer to sign, and the data that authenticates the signer). Communicates with the SSA for the purpose of transferring the generated SAD to the SAM. Can be used alternatively to the SAK/OS.
SAK	proNEXT SAK	CC certified signature application component (CC certificate: [CCertSAK]). Is situated in the remote TSP protected environment. Can be used to generate process relevant data structures as the SAD. Verifies the certificates generated during key pair generation. Collects certificate information for this purpose, evaluates it, and generates reports based on the checks.

SAK/OS	proNEXT SAK / Operations	Software that is installed in the signer's environment. Can be an application accessed from a browser or a mobile device, for example. Participates in the signature activation protocol (SAP) and generates the SAD. Provides the link between the signer and the signature process (linking the document to be signed, the remote signature key used by the signer to sign, and the data that authenticates the signer). Communicates with the SSA for the purpose of transferring the generated SAD to the SAM. Can be used as an alternative to the Remote Signature API.
SAM	proNEXT Signature Activation Module	A control unit for the cryptographic module, which is also located together with it in the specially secured environment. It registers users, initiates the generation of signature keys, is responsible for executing the signature process, and verifies the SAD. The SAM provides its own. With the help of the and information stored there about signers and authentication factors, it is ensured that only the actual owner of a key can access it and thus use it for remote signing. The SAM further activates the signature key against the cryptographic module. The SAM consists of three modules: the SAM Service, the SAM Firmware and the SAM Management. The SAM firmware is integrated into the cryptographic module.
SSA	proNEXT Server Signing Application	Acts as a kind of proxy for the controlled addressing of the functionality of the SAM and provides via it an interface to the cryptographic module for generating, holding and using signature keys. All requests to the SAM by the SIC or SAK/OS or users of the SAM shall be received, pre screened, and routed appropriately by the SSA. Signatories shall successfully identify and authenticate themselves before the SSA permits any actions involving the SAM. The SSA may maintain signer authentication for a specified period of time and/or for a specified number of signatures. In addition, the SSA creates audit records and passes them to the Audit Manager to manage audit logs.

Table 1: Components of the Server Signing Architecture

4.3 Users

The following is a list and description of the personal users that are used in the remote signature process.

User	Description
Privileged User	User that performs administrative functions (e.g. creating Signers).
Signer	A natural or legal person who is authorized to commission or execute server signatures. Provides the SAD via the SIC or the SAK/OS and can sign DTBS/R(s) with its own signature key assigned to it in the cryptographic module. The simple user becomes the Signer as soon as he receives the Signer certificate belonging to the signature key.
Privileged User Admin / Administrator	Privileged User exclusively authorized to install, configure and maintain the SAM. This role is managed by the operating system of the server environment where the SAM is installed.
Privileged User Technical	Privileged user who is only authorized to commission or initiate the commissioning of server signatures from the SSA at the SAM as a deputy technical user.

Table 2: User of the Signature Activation Module

The Privileged User is used in particular for the usage scenarios (US1) Privileged User Creation, (US2) Signer Creation and (US3) Signer Maintenance.

The usage scenarios (US4) Key Pair Generation, (US5) Key Pair Deletion are assigned to the Privileged User and the Signer.

Only the Signer is authorized to commit and have server signatures created using usage scenario (US6) Signing.

The usage scenario is responsible for executing the usage scenario (US7) SAM Maintenance for the purpose of managing the SAM.

5 Server Signing Processes

5.1 Server Signing Processes: Privileged User Creation

The Privileged User Creation process is divided into the following sections:

- User Authentication
- Privileged User Creation

The Privileged User Creation process corresponds to the usage scenario (US1) Privileged User Creation and the operation Privileged_User_Creation assigned to the SAM.

Sequence diagram:

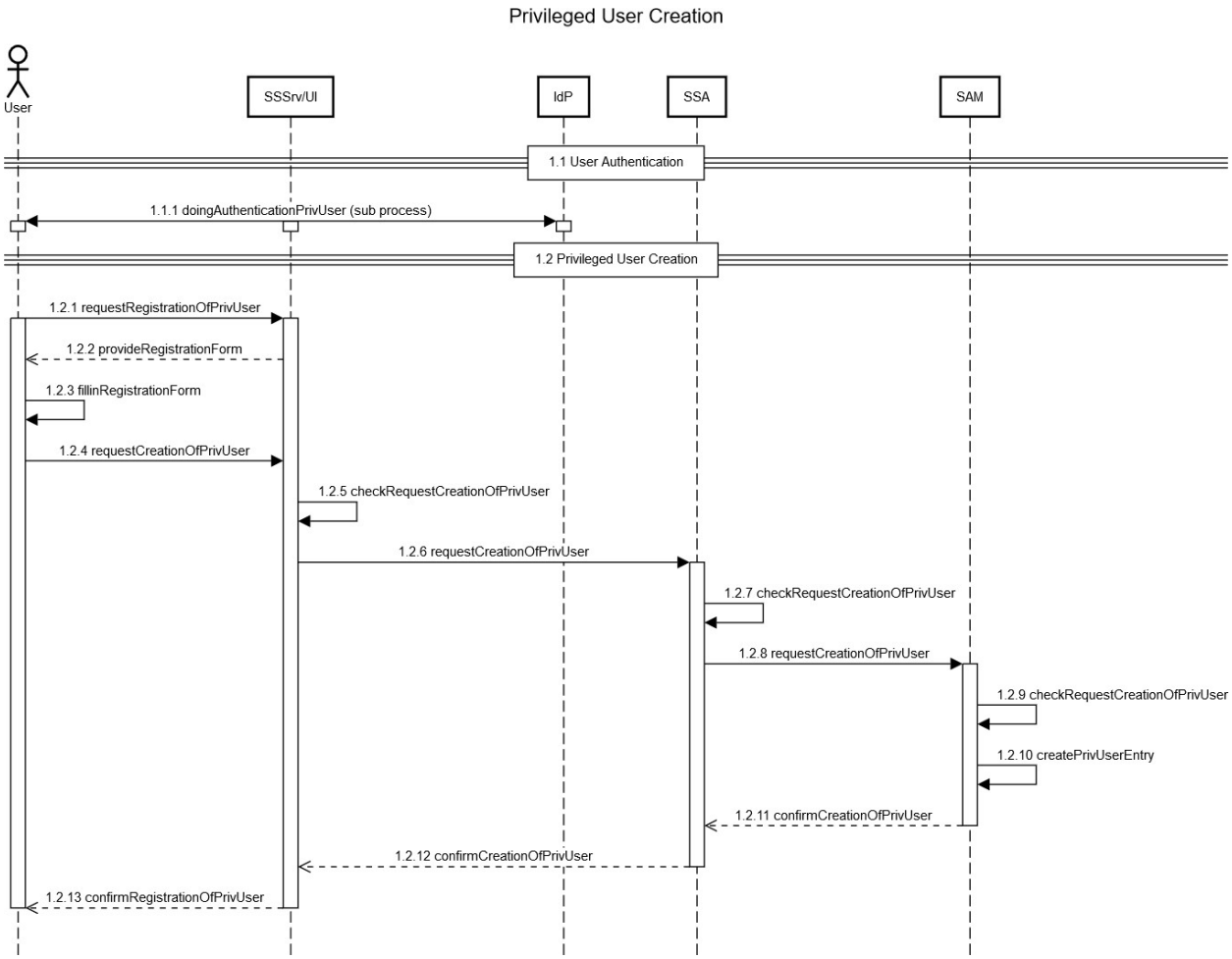


Figure 3: Sequence diagram of the Privileged User Creation process

Step-by-step description:

Nr.	Step	Components	Description
1 Privileged User Creation			
1.1 Privileged User Authentication			
1.1.1	The authentication of the Privileged User is performed.	User, SSSrv /UI, IdP	(11 doingAuthenticationPrivUser) Sub process according used means of identifications = Username/Password. Result is the ID Token for the Privileged User which signalize that the authentication is performed successfully.
1.2 Privileged User Creation			
1.2.1	The Privileged User requests the registration process for a new Privileged User at the SSSrv/UI.	User, SSSrv /UI	requestRegistrationOfPrivUser
1.2.2	The SSSrv/UI provides the registration form.	SSSrv/UI, User	provideRegistrationForm
1.2.3	The Privileged User fills in the registration form.	User	fillinRegistrationForm

1. 2. 4	The Privileged User requests the SSSrv/UI for the registration of a new Privileged User.	User, SSSrv /UI	requestCreationOfPrivUser
1. 2. 5	The SSSrv/UI checks the request for the registration of anew Privileged User.	SSSrv/UI	checkRequest CreationOfPrivUser
1. 2. 6	The SSSrv/UI requests the registration of a new Privileged User at the SSA.	SSSrv/UI, SSA	requestCreationOfPrivUser
1. 2. 7	The SSA checks the request for the registration of anew Privileged User.	SSA	checkRequest CreationOfPrivUser
1. 2. 8	The SSA requests the registration of a new Privileged User at the SAM.	SSA, SAM	requestCreationOfPrivUser
1. 2. 9	The SAM checks the request for the registration of a new Privileged User.	SAM	checkRequest CreationOfPrivUser
1. 2. 10	The UM creates the new entry to register the new Privileged User.	SAM	createPrivUserEntry

1. 2. 11	The SAM responses to the SSA confirming the creation of the new Privileged User.	SAM, SSA	confirmCreation OfPrivUser
1. 2. 12	The SSA responses to the SSSrv/UI confirming the creation of the new Privileged User.	SSA, SSSrv /UI	confirmCreation OfPrivUser
1. 2. 13	The SSSrv/UI responses the Privileged User to confirm the registration of a new Privileged User.	SSSrv/UI, User	confirmRegistration OfPrivUser

Table 3: Step-by-step description of the process Privileged User Creation

5.1.1 doingAuthenticationPrivUser

This process is a sub process of Privileged User Creation and other processes that require the execution by a Privileged User. It takes place in the user identification part and includes the technical authentication process using the username/password authentication mechanism. The result of the authentication process is that the IdP is able to create an ID token for the Privileged User.

Sequence diagram:

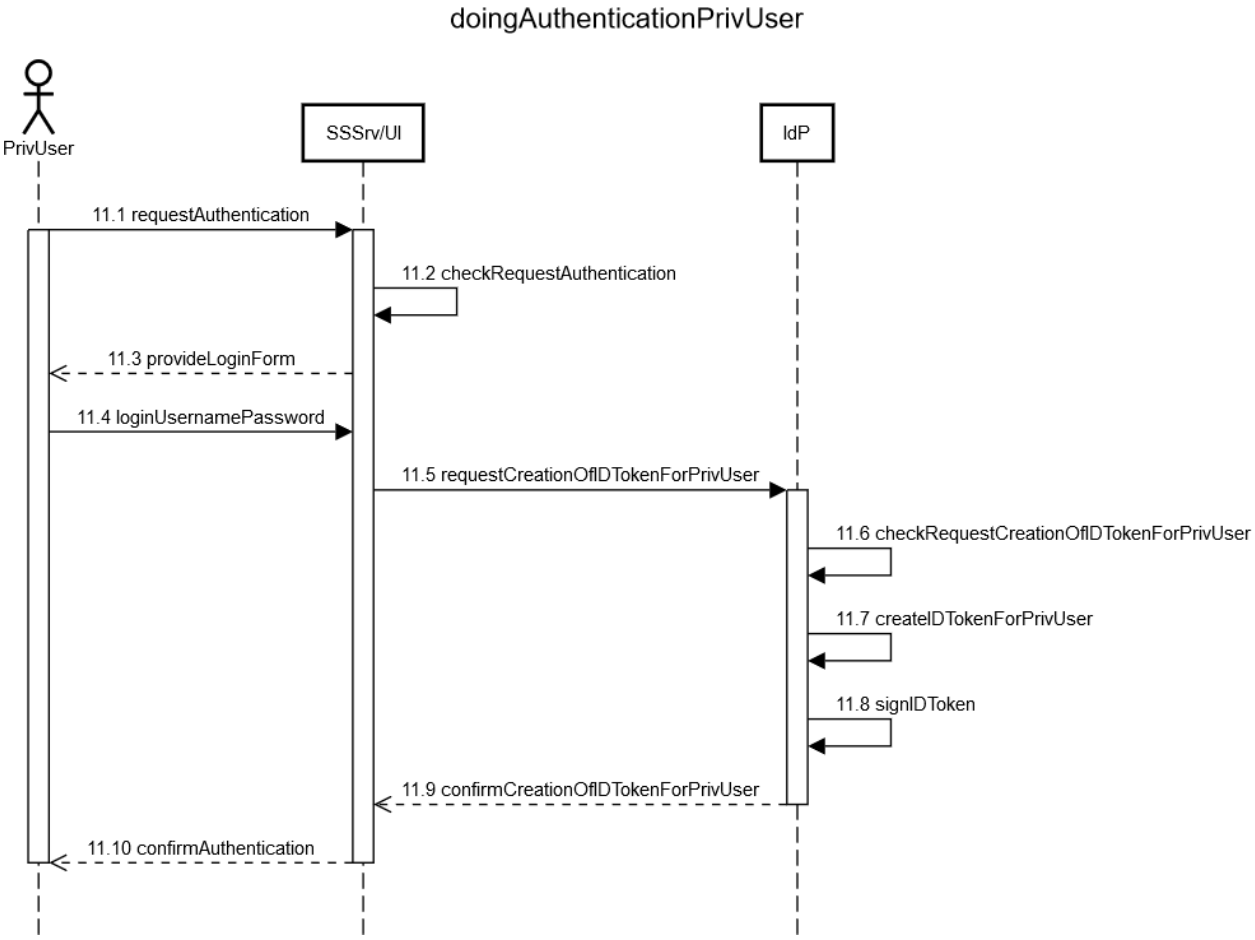


Figure 4: Sequence diagram of the doingAuthenticationPrivUser process

Step-by-step description:

Nr.	Step	Components	Description
11 doingAuthenticationPrivUser			
11.1	The Privileged User requests the login at the SSSrv /UI.	User, SSSrv /UI	requestAuthentication

11.2	The SSSrv/UI checks the request for the login.	SSSrv/UI	checkRequestAuthentication
11.3	The SSSrv/UI provides the login form to the Privileged User.	SSSrv/UI, User	provideLoginForm
11.4	The Privileged User logs on to the Privileged User Management.	User	loginUsernamePassword
11.5	The Privileged User requests the IdP for authentication.	User, IdP	requestCreationOfIDTokenForPrivUser
11.6	The IdP checks the requests for authentication.	IdP	checkRequestCreationOfIDTokenForPrivUser
11.7	The IdP generates a ID Token for the Privileged User.	IdP	createIDTokenForPrivUser
11.8	The IdP signs the generated ID Token.	IdP	signIDToken
11.9	The IdP responses the SSSrv/UI confirming the creation of the ID Token.	IdP, SSSrv/UI	confirmCreationOfIDTokenForPrivUser
11.10	The SSSrv/UI responses the Privileged User confirming the authentication by handing over the ID Token.	SSSrv/UI, User	confirmAuthentication

Table 4: Step-by-step description of the process doingAuthenticationPrivUser

5.2 Server Signing Processes: Signer Creation

The Signer Creation process is divided into the following sections: User Identification

Signer Creation

Corresponds to the usage scenario (US2) Signer Creation, includes among others the key pair generation.

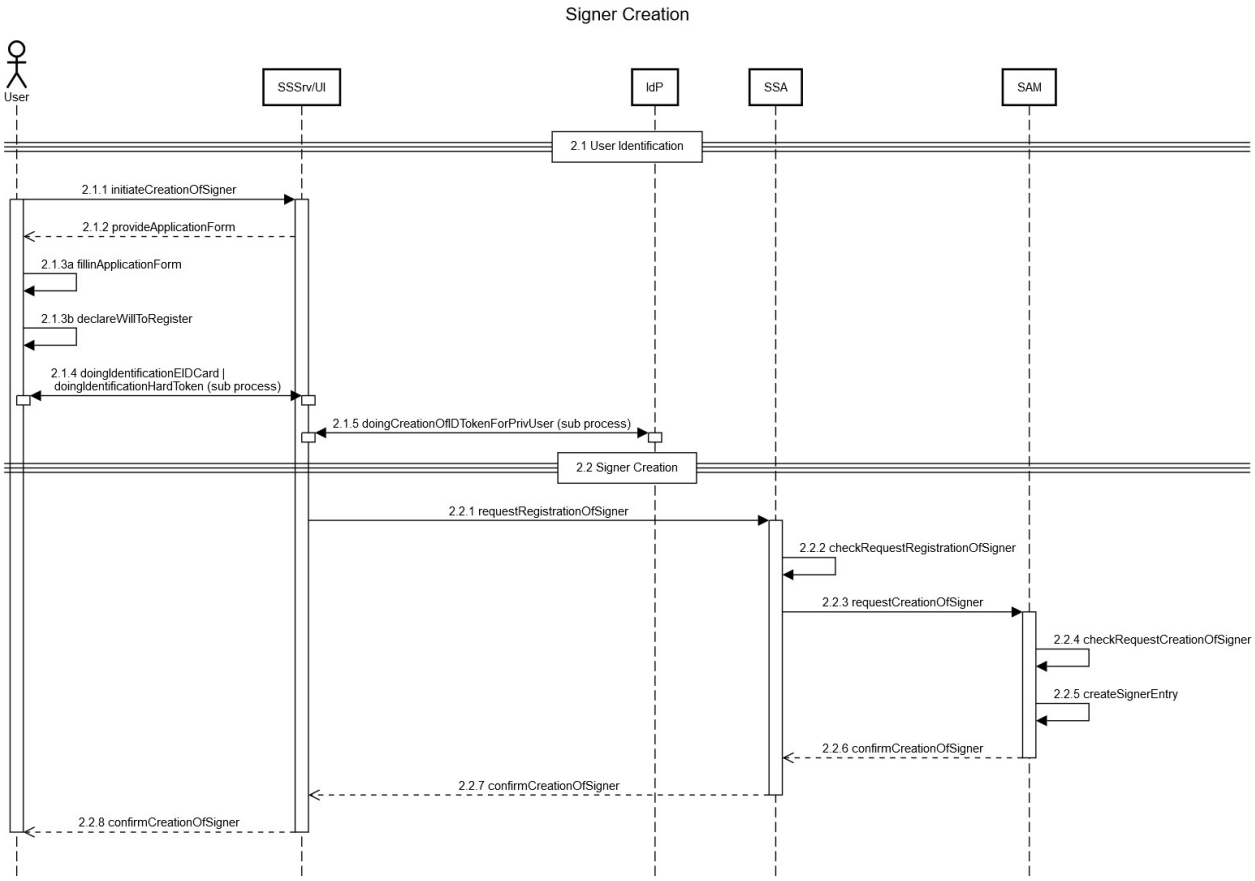


Figure 5: Sequence diagram of the Signer Creation process

Nr.	Step	Components	Description
2 Signer Creation			
2.1 User Identification			

2.1.1	The User initiates the process of being registered as Signer at the Server Signing Service by sending a request to the SSSrv/UI.	User, SSSrv /UI	initiateCreationOfSigner At this point the User starts performing either the use case 'New User' (eID Card) or the use case 'Existing User' (Hardware Token) and thus chooses the means of identifications used.
2.1.2	The SSSrv/UI provides to the User the appropriate application form for registration with the Server Signing Service.	SSSrv/UI, User	provideApplicationForm The application form for registration is presented by the SSSrv/UI via web interface according to the performed use case.
2.1.3	The User: fills in the application form activates the checkbox with which he declares his will for registration and reading of his personal data from the respective means of identification	User	fillinApplicationForm, declareWillToRegister Possible means of identification: eID Card, Hardware Token.

2.1.4	The identification of the User is performed.	User, SSSrv /UI	(21 doingIdentificationEIDCard | 22 doingIdentificationHardToken) Sub process according to the performed use case and the used means of identifications. Input is the application form data. Results are the Application Data Record and the storage of it.
2.1.5	The creation of an ID token for the Privileged User is performed.	SSSrv/UI, IdP	(23 doingCreation OfIDTokenForPrivUser)
2.2 Signer Creation			
2.2.1	The SSSrv/UI requests the SSA for the registration of a new Signer based on the determined and verified user data.	SSSrv/UI, SSA	requestRegistrationOfSigner

2.2.2	The SSA checks the request for the registration of a new Signer.	SSA	checkRequestRegistrationOfSigner The request shall contain the signed data structure which is generated in step before.
2.2.3	The SSA forwards the request to the SAM.	SSA, SAM	requestCreationOfSigner
2.2.4	The SAM checks the request for the creation of a Signer.	SAM	checkRequestCreationOfSigner
2.2.5	The SAM creates a new Signer entry.	SAM	createSignerEntry
2.2.6	The SAM responds to the SSA to confirm the creation of an UM entry for the Signer.	SAM, SSA	confirmCreationOfSigner
2.2.7	The SSAI responds to the SSSrv/UI to confirm the creation of the Signer.	SSA, SSSrv /UI	confirmCreationOfSigner

2.2.8	The SSSrv/UI responds to the User to confirm the creation of the Signer.	SSSrv/UI, User	confirmCreationOfSigner

Table 5: Step-by-step description of the process Signer Creation

5.2.1 doingIdentificationEIDCard

This process is a sub process of the Signer Creation. It takes place in the user identification part and includes the technical identification processes using the eID card as the means of identification. Input is the data of the application form. Result is the application record and the storage of it.

Sequence diagram

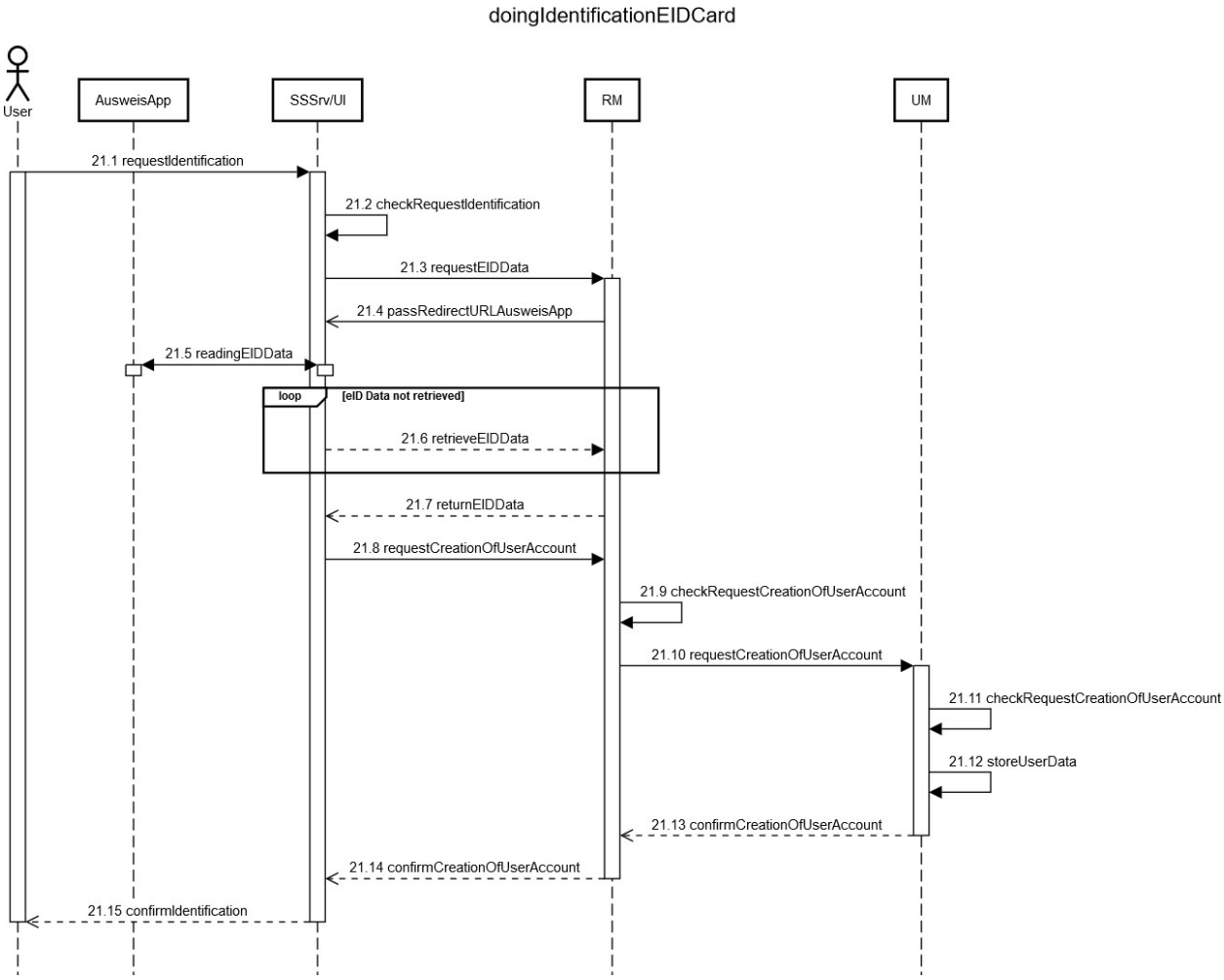


Figure 6: Sequence diagram of the doingIdentificationEIDCard process

Nr.	Step	Components	Description
21 doingIdentificationEIDCard			
21.1	The User requests the identification by handing over the data of the application form to the SSSrv/UI.	User, SSSrv /UI	requestIdentification
21.2	The SSSrv/UI checks the request for identification.	SSSrv/UI	checkRequestIdentification

21.3	The SSSrv/UI reads out the user data from the eID card used by the User by performing an eID process.	SSSrv/UI, RM	requestEIDData At this point of the process the User gives in his PIN according to the used eID Card (two factor authentication).
21.4	The RM passes the redirect URL used by the AusweisApp to perform the eID process to the SSSrv/UI.	RM, SSSrv/UI	passRedirectURLAusweisApp
21.5	The SSSrv/UI performs the eID process and reads the data of the users eID card.	SSSrv/UI, AusweisApp	readingEIDData At this point of the process the SSSrv/UI reads the data of the Users eID card per forming the eID process communicating with the AusweisApp.
21.6	The SSSrv/UI periodically requests to retrieve the read out eID data from the RM until the eID data is actually retrieved.	SSSrv/UI, RM	retrieveEIDData
21.7	The RM returns the read out eID data to the SSSrv/UI.	RM, SSSrv/UI	returnEIDData
21.8	The SSSrv/UI requests the creation of a new user account at the RM.	SSSrv/UI, RM	requestCreationOfUserAccount
21.9	The RM checks the request for creating of a new user account.	RM	checkRequestCreationOfUserAccount

21.10	The RM requests the creation of a new user account at the UM.	RM, UM	requestCreationOfUserAccount
21.11	The UM checks the request for creating of a new user account.	UM	checkRequestCreationOfUserAccount
21.12	The UM creates an new user account by storing the user data.	UM	storeUserData
21.13	The UM responds the RM to confirm the creation of a new user account.	UM, RM	confirmCreationOfUserAccount
21.14	The RM responds the SSSrv/UI to confirm the creation of a new user account.	RM, SSSrv /UI	confirmCreationOfUserAccount
21.15	The SSSrv/UI responds the User to confirm the performed identification.	SSSrv/ UI, User	confirmIdentification

Table 6: Step-by-step description of the process doingIdentificationEIDCard

5.2.2 doingIdentificationHardToken

This process is a sub process of the Signer Creation. It takes place in the user identification part and includes the technical identification processes using the hardware token as the means of identification. Input is the data of the application form. Result is the application record and the storage of it.

Sequence diagram

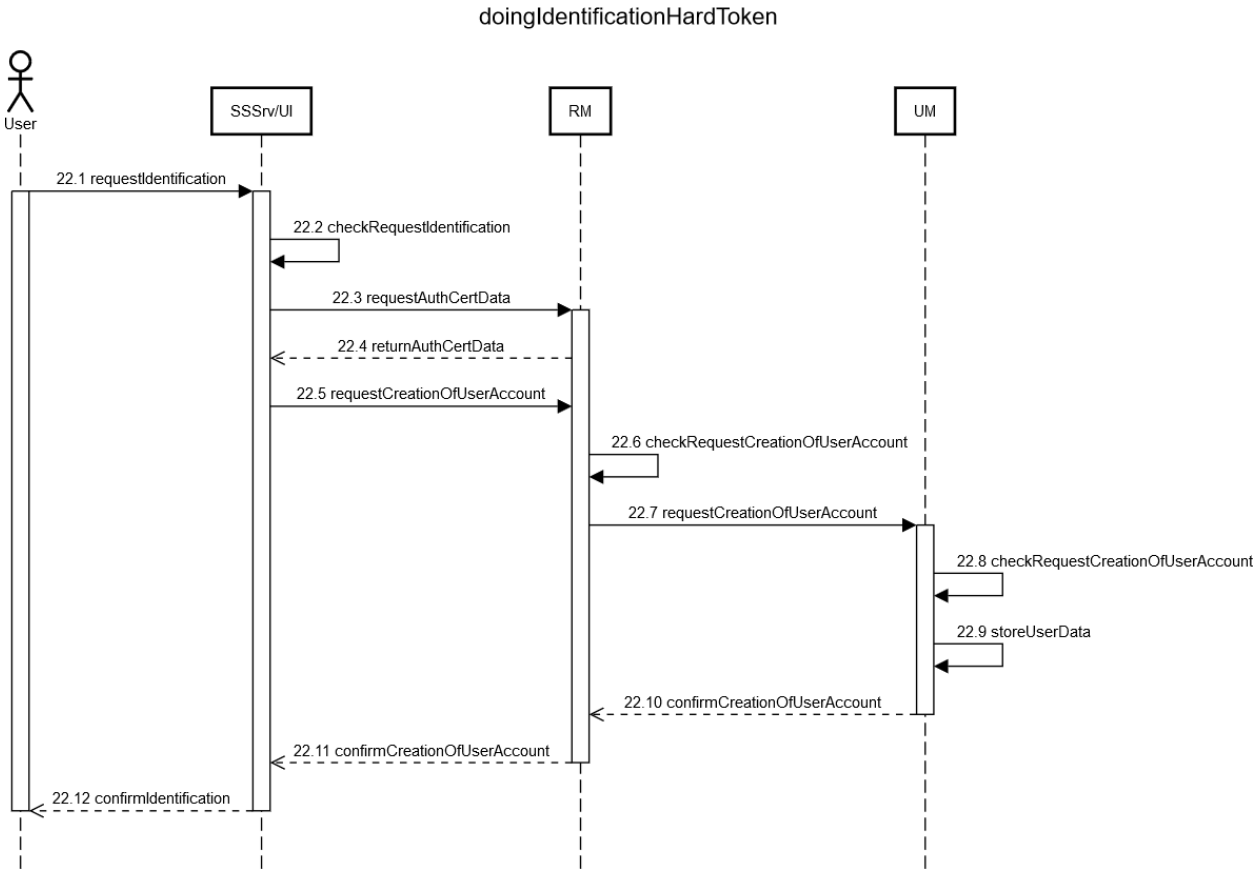


Figure 7: Sequence diagram of the doingIdentificationHardToken process

Nr.	Step	Components	Description
22 doingIdentificationHardToken			
22.1.	The User requests the identification by handing over the data of the application form to the SSSrv/UI.	User, SSSrv/UI	requestIdentification
22.2	The SSSrv/UI checks the request for identification.	SSSrv/UI	checkRequestIdentification

22.3	The SSSrv/UI reads out the user data from certificate provided by the User.	SSSrv/UI, RM	requestAuthCertData At this point of the process the User gives in his PIN according to the used eID Card (two factor authentication)
22.4	The RM returns the read out AuthCert datato the SSSrv/UI.	RM, SSSrv/UI	returnAuthCertData
22.5	The SSSrv/UI requests the creation of a new user account at the RM.	SSSrv/UI, RM	requestCreationOfUserAccount
22.6	The RM checks the request for creating of a new user account.	RM	checkRequestCreationOfUserAccount
22.7	The RM requests the creation of a new user account at the UM.	RM, UM	requestCreationOfUserAccount
22.8	The UM checks the request for creating of a new user account.	UM	checkRequestCreationOfUserAccount
22.9	The UM creates an new user account by storing the user data.	UM	storeUserData
22.10	The UM responds the RM to confirm the creation of a new user account.	UM, RM	confirmCreationOfUserAccount
22.11	The RM responds the SSSrv/UI to confirm the creation of a new user account.	RM, SSSrv/UI	confirmCreationOfUserAccount

22.12	The SSSrv/UI responds the User to confirm the performed identification.	SSSrv/UI, User	confirmIdentification

Table 7: Step-by-step description of the process doingIdentificationHardToken

5.2.3 doingCreationOfIDTokenForPrivUser

This process is a sub process of the Signer Creation. It takes place in the user identification part and involves requesting an ID token for the Privileged User in order to perform the operations assigned to the Privileged User. The result is that an ID token is issued by the IdP to the requesting entity.

Sequence diagram

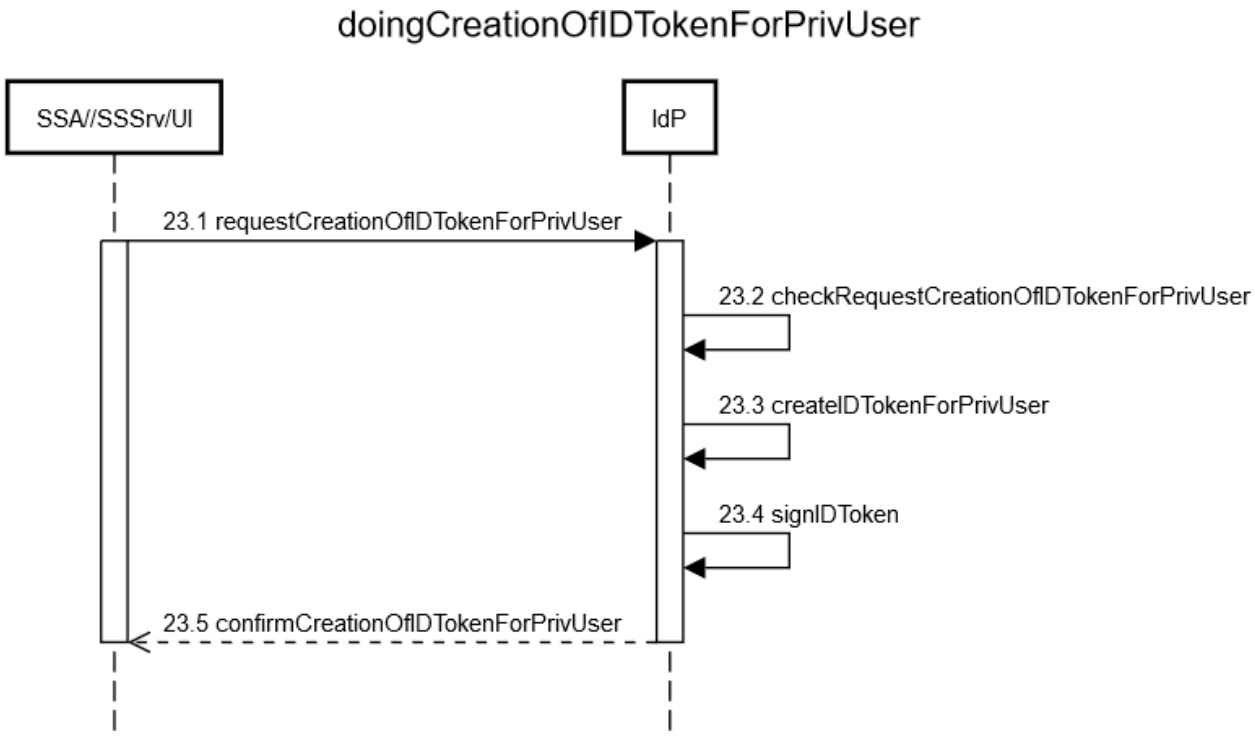


Figure 8: Sequence diagram of the doingCreationOfIDTokenForPrivUser process

Nr.	Step	Components	Description

23 doingCreationOfIDTokenForPrivUser			
23.1	The SSA//SSSrv/UI requests the IdP for the creation of an ID Token.	SSA//SSSrv/UI, IdP	requestCreationOfIDTokenForPrivUser
23.2	The IdP checks the request for the creation of an ID Token.	IdP	checkRequestCreationOfIDTokenForPrivUser
23.3	The IdP creates the requested ID Token for the Privileged User.	IdP	createIDTokenForPrivUser
23.4	The IdP signs the created ID Token.	IdP	signIDToken
23.5	The IdP responses the SSA//SSSrv/UI by handing over the ID Token.	IdP, SSA //SSSrv/UI	confirmCreationOfIDTokenForPrivUser

Table 8: Step-by-step description of the process doingCreationOfIDTokenForPrivUser

5.3 Server Signing Processes: Signer Maintenance

The Signer Maintenance process is divided into the following sections:

- User Authentication
- Signer Selection (optional)
- Signer Attributes Update

The Signer Maintenance process corresponds to the usage scenario (US3) Signer Maintenance and the operation Signer_Maintenance assigned to the SAM.

Sequence diagram

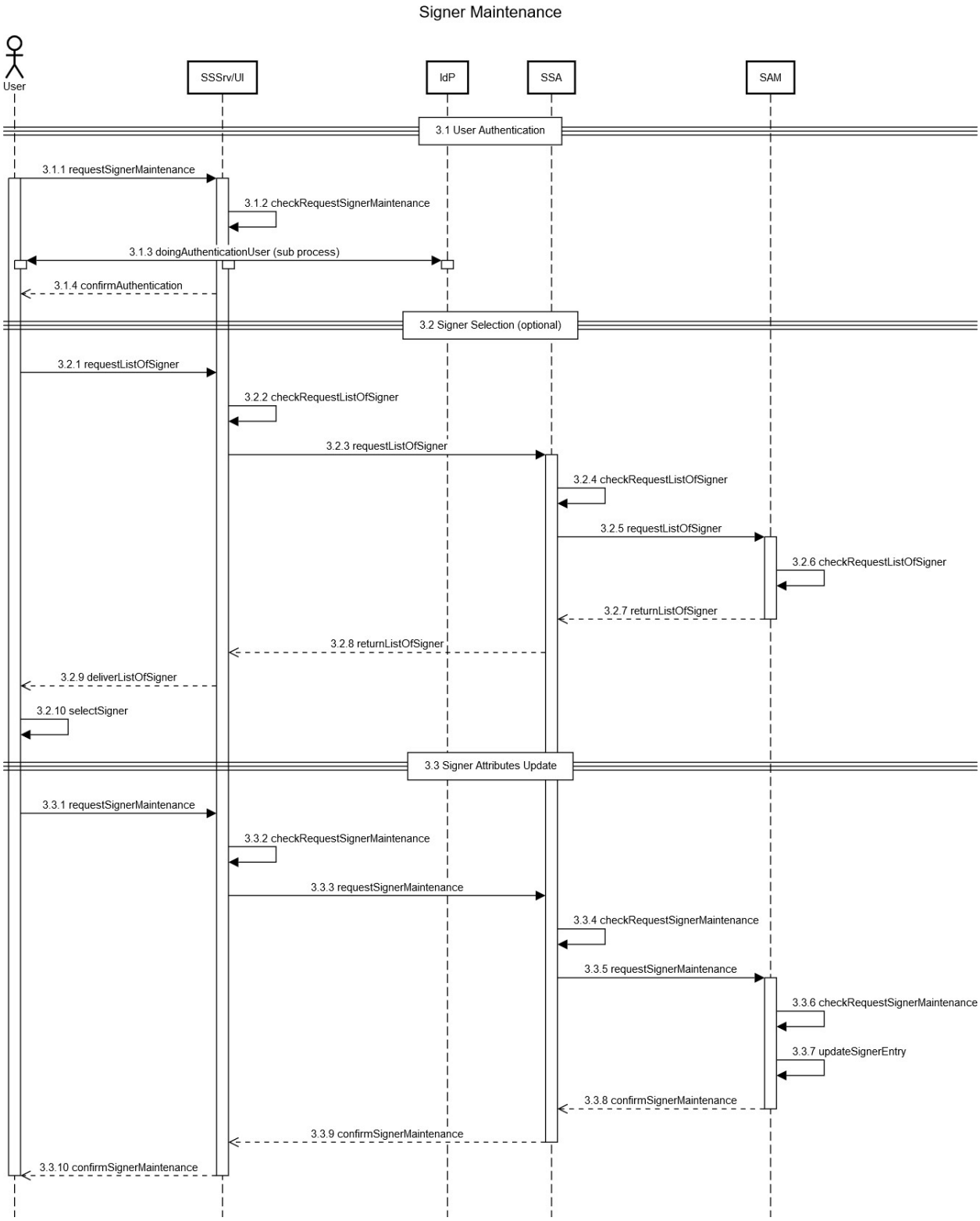


Figure 9: Sequence diagram of the Signer Maintenance process

Nr.	Step	Components	Description
3 Signer Maintenance			
3.1 User Authentication			
3.1.1	The User requests the Signer Maintenance at the SSSrv/UI.	User, SSSrv /UI	requestSignerMaintenance
3.1.2	The SSSrv/UI performs checks on the request for Signer Maintenance.	SSSrv/UI	checkRequest SignerMaintenance
3.1.3	The authentication of the User is performed.	User, SSSrv /UI, IdP	doingAuthenticationUser (11 doingAuthenticationPrivUser | 61 doingAuthenticationSigner) Sub process according to the used means of identification. Result is the ID Token for the User which signalize that the authentication is performed successfully.
3.1.4	The SSSrv/UI responses the User to confirm the authentication by transmitting the signed ID Token.	SSSrv/UI, User	confirmAuthentication
3.2 Signer Selection (optional)			

3.2.1	The Privileged User requests the list of Signer from the SSSrv/UI.	User, SSSrv /UI	requestListOfSigner
3.2.2	The SSSrv/UI performs checks on the request for the list of Signer.	SSSrv/UI	checkRequestListOfSigner
3.2.3	The SSSrv/UI requests the SSA for the list of Signer.	SSSrv/UI, SSA	requestListOfSigner
3.2.4	The SSA checks the request for the Signer list.	SSA	checkRequestListOfSigner
3.2.5	The SSA requests the list Signer.	SSA	requestListOfSigner
3.2.6	The SAM checks the request for the Signer list.	SAM	checkRequestListOfSigner
3.2.7	The SAM delivers the Signer list to the SSA.	SAM, SSA	returnListOfSigner
3.2.8	The SSA delivers the Signer list to the SSSrv/UI.	SSA, SSSrv /UI	returnListOfSigner

3.2.9	The SSSrv/UI delivers the Signer list to the Privileged User.	SSSrv/UI, User	deliverListOfSigner
3.2.10	The Privileged User selects the Signer.	User	selectSigner
3.3 Signer Attributes Update			
3.3.1	The User requests the update of his attributes at the SSSrv/UI.	User, SSSrv /UI	requestSignerMaintenance
3.3.2	The SSSrv/UI checks the request for the update of Signer attributes.	SSSrv/UI	checkRequestSignerMaintenance
3.3.3	The SSSrv/UI requests the update of Signer attributes at the SSA.	SSSrv/UI, SSA	requestSignerMaintenance
3.3.4	The SSA checks the request for the update of Signer attributes.	SSA	checkRequestSignerMaintenance
3.3.5	The SSA requests the update of Signer attributes at the SAM	SSA, SAM	requestSignerMaintenance

3.3.6	The SAM checks the request for the update of Signer attributes.	SAM	checkRequestSignerMaintenance
3.3.7	The SAM updates the entry of the Signer.	SAM	updateSignerEntry
3.3.8	The SAM responses the SSA confirming the update of the User attributes.	SAM, SSA	confirmSignerMaintenance
3.3.9	The SSA responses the SSSrv/UI confirming the update of the Signer attributes.	SSA, SSSrv /UI	confirmSignerMaintenance
3.3.10	The SSSrv/UI responses the User confirming the update of the Signer attributes.	SSSrv/UI, User	confirmSignerMaintenance

Table 9: Step-by-step description of the process Signer Maintenance

5.4 Server Signing Processes: Key Pair Generation

The Key Pair Generation process is divided into the following sections:

- User Authentication
- Signer Selection (optional)
- Signer Key Pair Generation
- Signer Certificate Issuance

The Key Pair Generation process corresponds to the usage scenario (US4) Key Pair Generation and the operation Signer_Key_Pair_Generation assigned to the SAM.

Sequence diagram

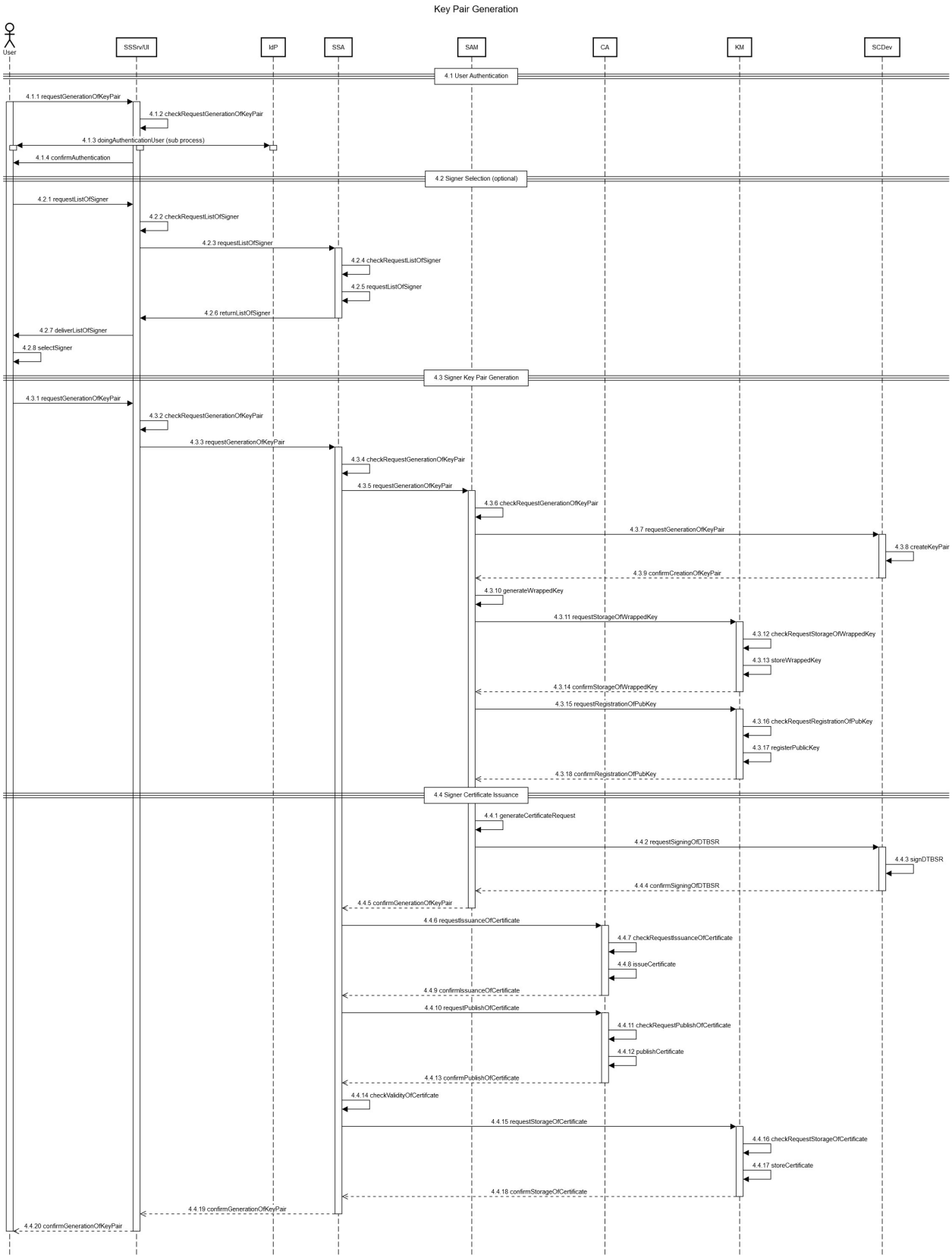


Figure 10: Sequence diagram of the Key Pair Generation process

Nr.	Step	Components	Description
4 Key Pair Generation			
4.1 User Authentication			
4.1.1	The User requests the SSSrv/UI for Key Pair Generation.	User, SSSrv /UI	requestGenerationOfKeyPair
4.1.2	The SSSrv/UI performs checks on the request for Key Pair Generation.	SSSrv/UI	checkGenerationOfKeyPair
4.1.3	The authentication of the User is performed.	User, SSSrv /UI, IdP	doingAuthenticationUser (11 doingAuthenticationPrivUser | 61 doingAuthenticationSigner) Sub process according used means of identifications = Username/Password. Result is the ID Token for the User which signalize that the authentication is performed successfully.
4.1.4	The SSSrv/UI responses the User to confirm the authentication by transmitting the signed ID Token.	SSSrv/UI, User	confirmAuthentication

4.2 Signer Selection (optional)			
4.2.1	The Privileged User requests the list of Signer from the SSSrv/UI.	User, SSSrv /UI	requestListOfSigner
4.2.2	The SSSrv/UI performs checks on the request for the list of Signer.	SSSrv/UI	checkRequestListOfSigner
4.2.3	The SSSrv/UI requests the SSA for the list of Signer.	SSSrv/UI, SSA	requestListOfSigner
4.2.4	The SSA checks the request for the Signer list.	SSA	checkRequestListOfSigner
4.2.5	The SSA requests the list of Signer.	SSA	requestListOfSigner
4.2.6	The SSA delivers the Signer list to the SSSrv/UI.	SSA, SSSrv /UI	returnListOfSigner
4.2.7	The SSSrv/UI delivers the Signer list to the Privileged User.	SSSrv/UI, User	deliverListOfSigner

4.2.8	The Privileged User selects the Signer.	User	selectSigner
4.4.1	The SAM generates a certificate request based on the generated key material.	SAM	generateCertificateRequest
4.4.2	The SAM requests the SCDev to sign the supplied DTBS/R.	SAM, SCDev	requestSigningOfDTBSR
4.4.3	The SCDev signs the supplied DTBS/R.	SCDev	signDTBSR
4.4.4	The SCDev responses the SAM confirming the signing of the DTBS /R by returning the signed DTBS/R.	SCDev, SAM	confirmSigningOfDTBSR
4.4.5	The SAM confirms the creation of a key pair to the SSA.	SAM, SSA	confirmGenerationOfKeyPair
4.4.6	The SSA requests the issuance of a X.509 certificate based on the certificate at the CA.	SSA, CA	requestIssuanceOfCertificate

4.4.7	The CA performs checks on the request for issuing a certificate.	CA	checkRequest IssuanceOfCertificate
4.4.8	The CA issues the requested Certificate.	CA	issueCertificate
4.4.9	The CA responds the SSA to confirm the issuance of the certificate by handing over the issued certificate to the SSA.	CA, SSA	confirmIssuanceOfCertificate
4.4.10	The SSA requests the publication of the issued certificate at the CA.	SSA, CA	requestPublicationOfCertificate
4.4.11	The CA performs checks on the request for publishing a certificate.	CA	checkRequest PublicationOfCertificate
4.4.12	The CA publish the provided Certificate.	CA	publishCertificate
4.4.13	The CA respond the SSA to confirm the publication of the certificate.	CA, SSA	confirmPublicationeOfCertificate
4.4.14	The SSA checks the validity of the certificate.	SSA	checkValidityOfCertifcate

4.4.15	The SSA requests the KM to store the certificate for the specified User.	SSA, KM	requestStorageOfCertificate
4.4.16	The KM checks the request for the storage of the certificate.	KM	checkRequestStorageOfCertificate
4.4.17	The KM stores the certificate for the specified User.	KM	storeCertificate
4.4.18	The KM responds to the SSA to confirm the storage of the certificate.	KM, SSA	confirmStorageOfCertificate
4.4.19	The SSA responses to the SAK/OS to confirm the registration of the user.	SSA, SSSrv /UI	confirmRegistrationOfUser
4.3 Signer Key Pair Generation			
4.3.1	The User requests the SSSrv/ UI to generate new key pair.	SSA, SSSrv /UI	requestGenerationOfKeyPair
4.3.2	The SSSrv/UI checks the request for generating a new key pair.	SSSrv/UI	checkRequest GenerationOfKeyPair

4.3.3	The SSSrv/UI requests the SSA to generate new key pair.	SSA, SSA	requestGenerationOfKeyPair
4.3.4	The SSA checks the request for generating a new key pair.	SSA	checkRequest GenerationOfKeyPair
4.3.5	The SSA requests the SAM to generate new key pair.	SSA, SAM	requestGenerationOfKeyPair
4.3.6	The SAM checks the request for generating a new key pair.	SAM	checkRequest GenerationOfKeyPair
4.3.7	The SAM requests the SCDev to generate new key pair.	SAM, SCDev	requestGenerationOfKeyPair
4.3.8	The SCDev creates a new key pair.	SCDev	createKeyPair
4.3.9	The SCDev confirms the creation of key pair by returning KeyID and the Public Key to the SAM.	SCDev, SAM	confirmCreationOfKeyPair
4.3.10	The SAM generates the Wrapped Key based on the generated key material and its properties.	SAM	generateWrappedKey

4.3.11	The SAM requests the KM for the storage of Wrapped Key.	SAM, KM	requestStorage OfWrappedKey
4.3.12	The KM performs checks on the request to store Wrapped Key.	KM	checkRequestStorage OfWrappedKey
4.3.13	The KM stores Wrapped Key including a user mapping.	KM	storeWrappedKey
4.3.14	The KM responds to the SAM confirming the storage of Wrapped Key.	KM, SAM	confirmStorage OfWrappedKey
4.3.15	The SAM requests the KM to register the public key of the generated key pair.	SAM, KM	requestRegistrationOfPublicKey
4.3.16	The KM checks the request for registration of the public key of the generated key pair	KM	checkRequestRegistrationOfPublicKey
4.3.17	The KM stores the registers the public key of the generated key pair	KM	registerPublicKey
4.3.18	The KM responds to the SAM confirming the registration of the public key of the generated key pair.	KM, SAM	confirmRegistrationOfPublicKey

4.4 Signer Certificate Issuance			
4.4.20	The SSA responses to the SAK/OS to confirm the registration of the user.	SSSrv/UI, User	confirmRegistrationOfUser

Table 10: Step-by-step description of the process Key Pair Generation

5.5 Server Signing Processes: Key Pair Deletion

The Key Pair Deletion process is divided into the following sections:

- User Authentication
- Signer Selection (optional)
- Signing Key Selection
- Signing Key Pair Deletion

The Key Pair Deletion process corresponds to the usage scenario (US5) Key Pair Deletion and the operation Signer_Key_Pair_Generation assigned to the SAM.

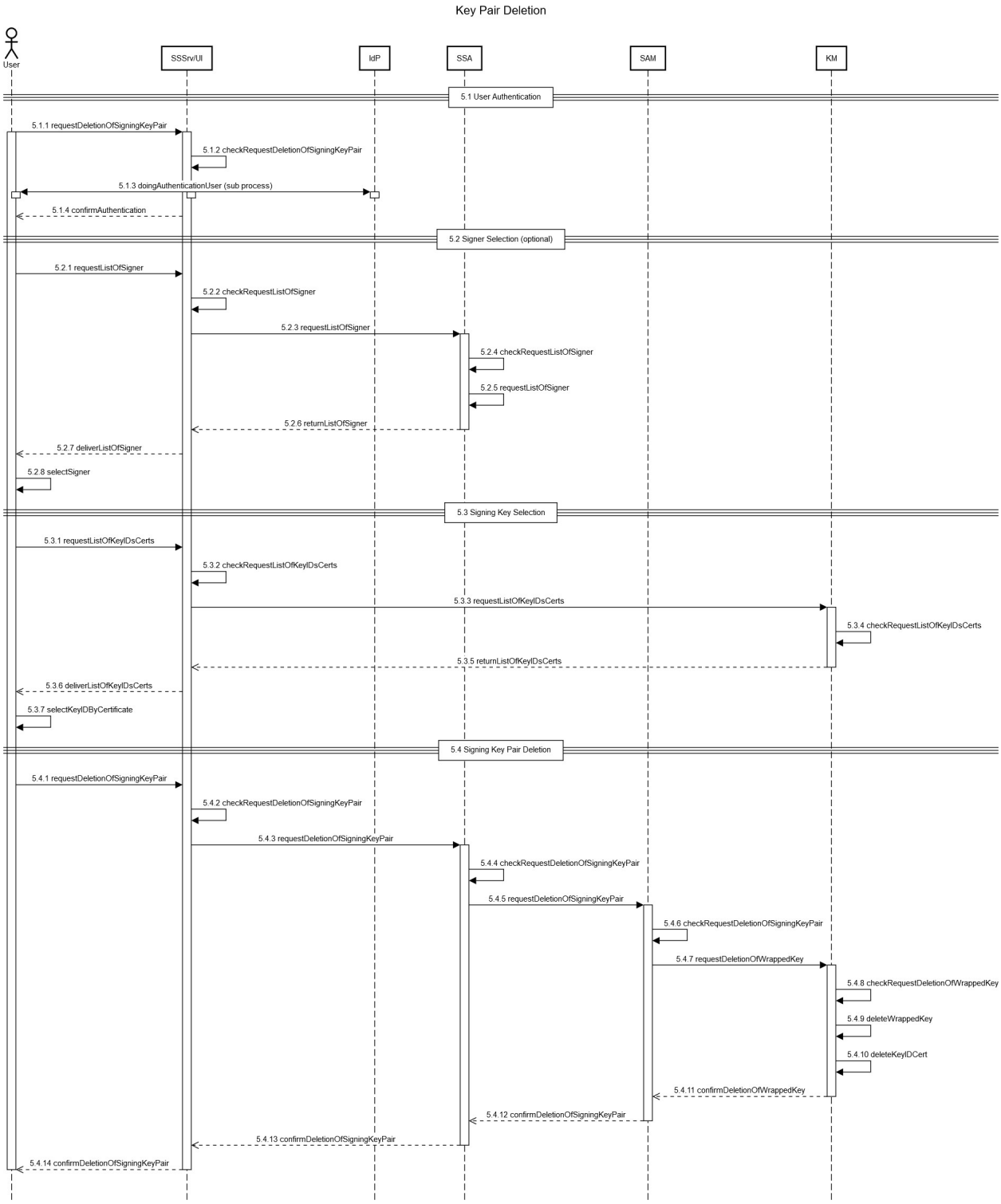


Figure 11: Sequence diagram of the Key Pair Deletion process

Nr.	Step	Components	Description
5 Key Pair Deletion			
5.1 User Authentication			
5.1.1	The User requests the Signer Key Pair Deletion at the SSSrv/UI.	User, SSSrv /UI	requestSignerKeyPairDeletion
5.1.2	The SSSrv/UI performs checks on the request for Signer Maintenance.	SSSrv/UI	checkRequest SignerKeyPairDeletion
5.1.3	The authentication of the User is performed.	User, SSSrv /UI, IdP	doingAuthenticationUser (11 doingAuthenticationPrivUser | 61 doingAuthenticationSigner) Sub process according to the used means of identification. Result is the ID Token for the User which signalize that the authentication is performed successfully.
5.1.4	The SSSrv/UI responses the User to confirm the authentication by transmitting the signed ID Token.	SSSrv/UI, User	confirmAuthentication
5.2 Signer Selection (optional)			

5.2.1	The Privileged User requests the list of Signer from the SSSrv/UI.	User, SSSrv /UI	requestListOfSigner
5.2.2	The SSSrv/UI performs checks on the request for the list of Signer.	SSSrv/UI	checkRequestListOfSigner
5.2.3	The SSSrv/UI requests the SSA for the list of Signer.	SSSrv/UI, SSA	requestListOfSigner
5.2.4	The SSA checks the request for the Signer list.	SSA	checkRequestListOfSigner
5.2.5	The SSA requests the list of Signer.	SSA	requestListOfSigner
5.2.6	The SSA delivers the Signer list to the SSSrv/UI.	SSA, SSSrv /UI	returnListOfSigner
5.2.7	The SSSrv/UI delivers the Signer list to the Privileged User.	SSSrv/UI, User	deliverListOfSigner
5.2.8	The Privileged User selects the Signer.	User	selectSigner
5.3 Signing Key Selection			

5.3.1 (optional)	The User requests the list of KeyIDs and assigned certificates corresponding to the ID Token from the SSSrv/UI.	User, SSSrv /UI	requestListOfKeyIDsCerts
5.3.2 (optional)	The SSSrv/UI performs checks on the request for the list of KeyIDs and assigned certificates.	SSSrv/UI	checkRequest ListOfKeyIDsCerts
5.3.3 (optional)	The SSSrv/UI requests the KM for the KeyID list matching the transferred ID Token.	SSSrv/UI, KM	requestListOfKeyIDsCerts
5.3.4 (optional)	The KM checks the request for the list.	KM	checkRequest ListOfKeyIDsCerts

5.3.5 (optional)	The KM returns the KeyID list including corresponding certificate information to the SSSrv/UI.	KM, SSSrv /UI	returnListOfKeyIDsCerts
5.3.6 (optional)	The SSSRV/UI delivers the identified KeyIDs and related certificates to the User.	SSSrv/UI, User	deliverListOfKeyIDsCerts
5.3.7	The User selects the certificate and thus the associated KeyID to be deleted.	User	selectKeyIDByCertificate
5.4 Signing Key Pair Deletion			
5.4.1	The User requests the deletion of the Signing Key at the SSSrv/UI.	User, SSSrv /UI	requestDeletion OfSigningKeyPair
5.4.2	The SSSrv/UI checks the request for the deletion of the Signing Key.	SSSrv/UI	checkRequestDeletion OfSigningKeyPair
5.4.3	The SSSrv/UI requests the deletion of the Signing Key at the SSA.	SSSrv/UI, SSA	requestDeletion OfSigningKeyPair

5.4.4	The SSA checks the request for the deletion of the Signing Key.	SSA	checkRequestDeletion OfSigningKeyPair
5.4.5	The SSA requests the deletion of the Signing Key at the SAM.	SSA, SAM	requestDeletion OfSigningKeyPair
5.4.6	The SAM checks the request for the deletion of the Signing Key.	SAM	checkRequestDeletion OfSigningKeyPair
5.4.7	The SAM requests the deletion of the Wrapped Key at the KM.	SAM, KM	requestDeletion OfWrappedKey
5.4.8	The KM checks the request for the deletion of the Wrapped.	KM	checkRequestDeletion OfWrappedKey
5.4.9	The KM deletes the Wrapped Key.	KM	deleteWrappedKey
5.4.10	The KM deletes the KeyID certificate.	KM	deleteKeyIDCert
5.4.11	The UM responses the SAM to confirm the deletion of the Wrapped Key.	KM, SAM	confirmDeletionOfWrappedKey
5.4.12	The SAM responses the SSA confirming the deletion of the Signing Key Pair.	SAM, SSA	confirmDeletion OfSigningKeyPair

5.4.13	The SSA responses the SSSrv/UI confirming the deletion of the Signing Key Pair.	SSA, SSSrv /UI	confirmDeletion OfSigningKeyPair
5.4.14	The SSSrv/UI responses the User confirming the deletion of the Signing Key Pair.	SSSrv/UI, User	confirmDeletion OfSigningKeyPair

Table 11: Step-by-step description of the process Key Pair Deletion

5.6 Server Signing Processes: Signing

The signing process is divided into the following sections:

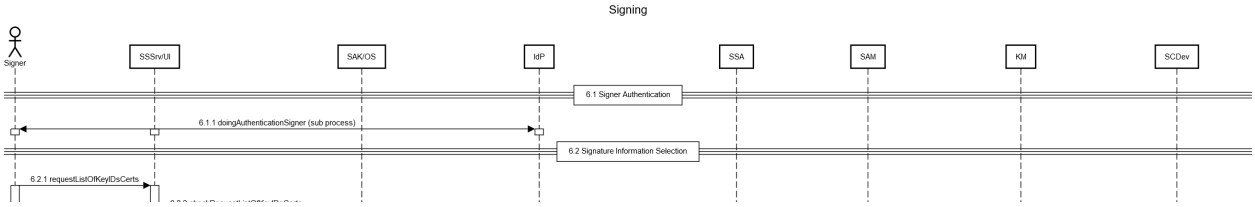
- Signer Authentication
- Signature Information Selection
- SAD Generation
- Signing Key Activation
- Signature Value Creation

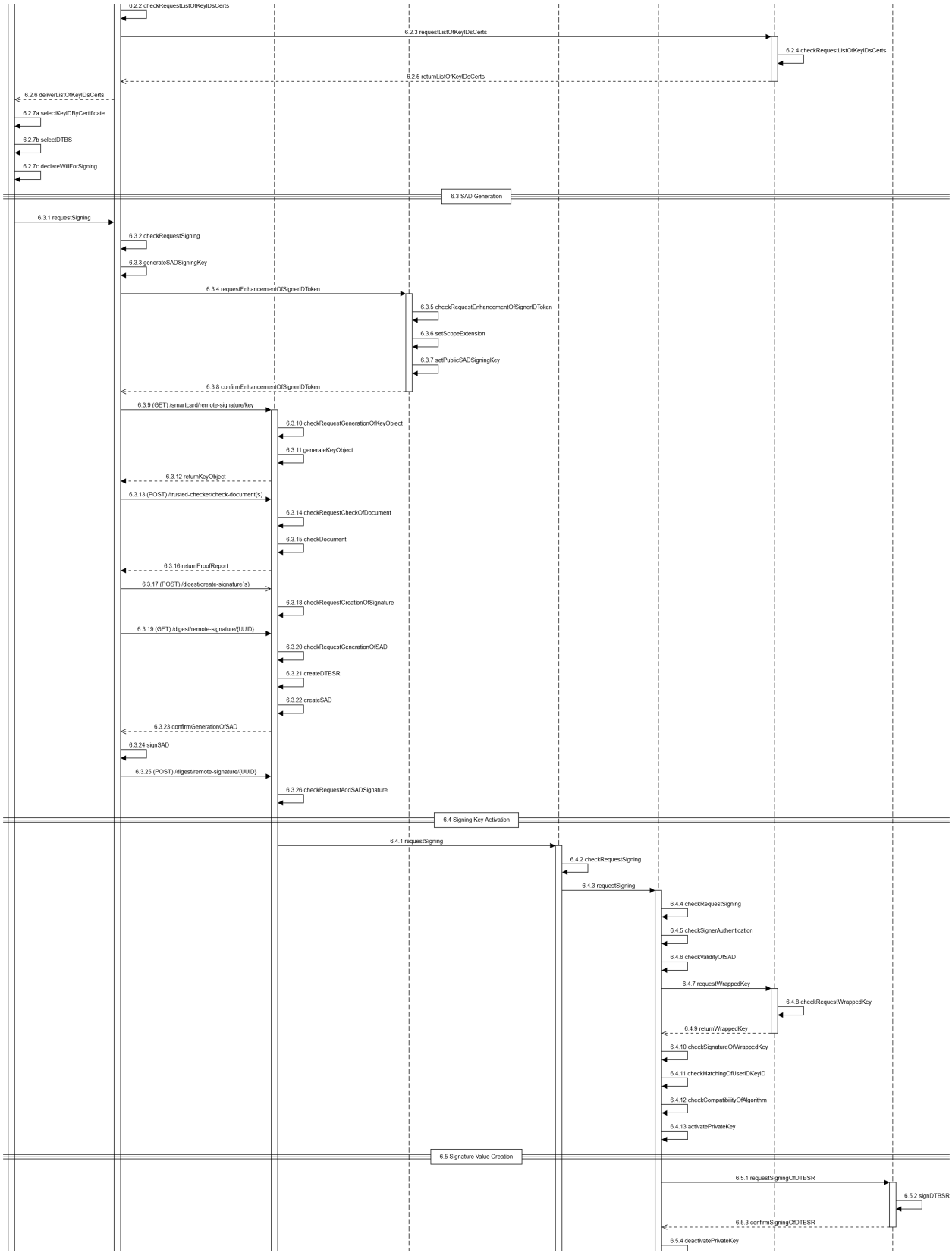
The Signing process corresponds to the usage scenario (US6) Signing and the operation Signing assigned to the SAM.

There are basically two different approaches to commissioning remote signature. The both resulting processes differ when it comes to the SAD generation. Either the SAD ist generated via a locally installed SAK/OS or this is done via a remote SAK. Both processes are described below.

5.6.1 Signing using local SAK/OS

Sequence diagram




6.2.2 checkRequestListOfKeyIDsCerts
6.2.3 requestListOfKeyIDsCerts
6.2.4 checkRequestListOfKeyIDsCerts
6.2.5 returnListOfKeyIDsCerts
6.2.6 deliverListOfKeyIDsCerts
6.2.7a selectKeyIDByCertificate
6.2.7b selectDTBS
6.2.7c declareWillForSigning
6.3 SAD Generation
6.3.1 requestSigning
6.3.2 checkRequestSigning
6.3.3 generateSADSigningKey
6.3.4 requestEnhancementOfSignerIDToken
6.3.5 checkRequestEnhancementOfSignerIDToken
6.3.6 setScopeExtension
6.3.7 setPublicSADSigningKey
6.3.8 confirmEnhancementOfSignerIDToken
6.3.9 (GET) /smartcard/remote-signature/key
6.3.10 checkRequestGenerationOfKeyObject
6.3.11 generateKeyObject
6.3.12 returnKeyObject
6.3.13 (POST) /trusted-checker/check-document(s)
6.3.14 checkRequestCheckOfDocument
6.3.15 checkDocument
6.3.16 returnProofReport
6.3.17 (POST) /digest/create-signature(s)
6.3.18 checkRequestCreationOfSignature
6.3.19 (GET) /digest/remote-signature/{UUID}
6.3.20 checkRequestGenerationOfSAD
6.3.21 createDTBSR
6.3.22 createSAD
6.3.23 confirmGenerationOfSAD
6.3.24 signSAD
6.3.25 (POST) /digest/remote-signature/{UUID}
6.3.26 checkRequestAddSADSignature
6.4 Signing Key Activation
6.4.1 requestSigning
6.4.2 checkRequestSigning
6.4.3 requestSigning
6.4.4 checkRequestSigning
6.4.5 checkSignerAuthentication
6.4.6 checkValidityOfSAD
6.4.7 requestWrappedKey
6.4.8 checkRequestWrappedKey
6.4.9 returnWrappedKey
6.4.10 checkSignatureOfWrappedKey
6.4.11 checkMatchingOfUserIDKeyID
6.4.12 checkCompatibilityOfAlgorithm
6.4.13 activatePrivateKey
6.5 Signature Value Creation
6.5.1 requestSigningOfDTBSR
6.5.2 signDTBSR
6.5.3 confirmSigningOfDTBSR
6.5.4 deactivatePrivateKey

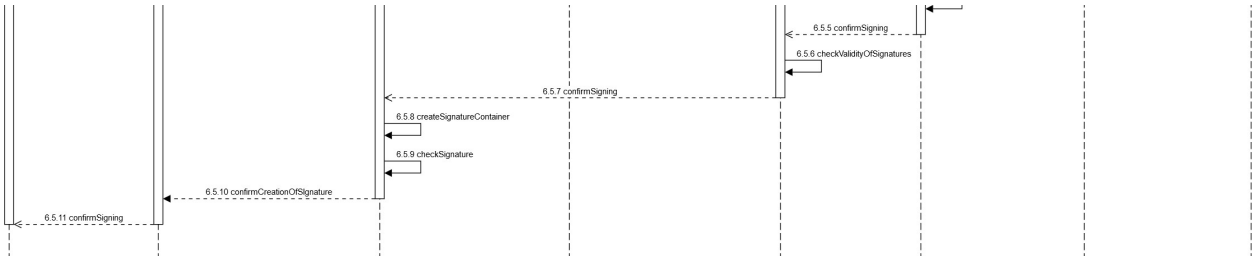


Figure 12: Sequence diagram of the Signing process

Step-by-step description

Nr.	Step	Components	Description
6 Signing			
6.1 Signer Authentication			
6.1.1	The Signer performs the authentication process.	Signer, SSSrv/UI, IdP	(61 doingAuthentificationSigner) Subprocess that authenticates the Signer based on the identification means used. The result is the ID token for the Signer, which signals that authentication has been successfully performed.
6.2 Signature Information Selection			
6.2.1 (optional)	The Signer requests the list of KeyIDs and associated certificates corresponding to the ID token from the SSSrv/UI.	Signer, SSSrv/UI	requestListOfKeyIDsCerts

6.2.2 (optional)	The SSSrv/UI checks the request for the list of KeyIDs and associated certificates.	SSSrv/UI	checkRequest ListOfKeyIDsCerts
6.2.3 (optional)	The SSSrv/UI queries the KM for the KeyID list that matches the specified ID token.	SSSrv/UI, KM	requestListOfKeyIDsCerts
6.2.4 (optional)	The KM checks the request for the list of KeyIDs.	KM	checkRequest ListOfKeyIDsCerts
6.2.5 (optional)	The KM sends the KeyID list with the corresponding certificate information back to the SSSrv/UI.	KM, SSSrv/UI	returnListOfKeyIDsCerts

6.2.6 (optional)	The SSSrv/UI delivers the identified KeyIDs and associated certificates to the Signer.	SSSrv/UI, Signer	deliverListOfKeyIDsCerts
6.2.7	The Signer selects the certificate and thus the associated KeyID to be used for signing, selects the data to be signed, and makes a declaration of intent to sign.	Signer	aselectKeyIDByCertificate selectDTBS declareWillForSigning The declaration of intent is realized via a checkbox. The SHA512 hash of the document and the certificate details of the certificate used for the signature are displayed to the user.
6.3 SAD Generation			
6.3.1	The undersigned requests the SSSrv/UI to remotely sign data.	Signer, SSSrv/UI	requestSigning
6.3.2	The SSSrv/UI checks the authorization regarding the request to sign.	SSSrv/UI	checkRequestSigning
6.3.3	The SSSrv/UI generates the temporary key for the later signature of the SAD as well as the embedding of the public part into the ID token of the Signer.	SSSrv/UI	generateSADSigningKey

6.3.4	The SSSrv/UI requests the extension of the Signer's ID token.	SSSrv/UI, IdP	requestEnhancementOfSignerIDToken
6.3.5	The IdP checks the requests for enrichment of the Signer's ID token.	IdP	checkRequestEnhancementOfSignerIDToken
6.3.6	The IdP extends the ID token by setting the scope extension.	IdP	setScopeExtension
6.3.7	The IdP extends the ID token by setting the public part of the SAD signature key pair.	IdP	setPublicSADSigningKey
6.3.8	The IdP confirms the extension of the Signer's ID token and sends the extended ID token to the SSSrv/UI.	IdP, SSSrv/UI	confirmEnhancementOfSignerIDToken
6.3.9	The SSSrv/UI requests the creation of a Key Object at the SAK/OS.	SSSrv/UI, SAK/OS	(GET) /smartcard/remote-signature/key
6.3.10	The SAK/OS checks the request to create a Key Object.	SAK/OS, SAK/OS	checkRequestGenerationOfKeyObject
6.3.11	The SAK/OS generates a Key Object.	SAK/OS, SAK/OS	generateKeyObject

6.3.12	The SAK/OS responds with the generated Key Object.	SAK/OS, SSSrv/UI	returnKeyObject
6.3.13	The SSSrv/UI requests review of the document to be signed by the SAK	SSSrv/UI, SAK/OS	(POST) /trusted-checker/check-document(s)
6.3.14	The SAK/OS will consider the request to review the document to be signed.	SAK/OS	checkRequestCheckOfDocument
6.3.15	The SAK/OS reviews the document.	SAK/OS	checkDocument
6.3.16	The SAK/OS responds with the audit report to the SSSrv/UI.	SAK/OS, SSSrv/UI	returnProofReport
6.3.17	The SSSrv/UI requests the creation of a (remote) signature at the SAK/OS.	SSSrv/UI, SAK/OS	(POST) /digest/create-signature(s)
6.3.18	The SAK/OS verifies the request to create a (remote) signature.	SAK/OS	checkRequestCreationOfSignature

6.3.19	The Signer requests the SAK/OS to generate a SAD with the information required for signing.	SSSrv/UI, SAK/OS	(GET) /digest/remote-signature/{UUID}
6.3.20	The SAK/OS checks the request to generate SAD.	SAK/OS	checkRequestGenerationOfSAD
6.3.21	The SAK/OS generates a DTBS /R from the DTBS.	SAK/OS	createDTBSR
6.3.22	The SAK/OS creates the SAD.	SAK/OS	createSAD
6.3.23	The SAK/OS confirms the generation of the SAD.	SAK/OS, SSSrv/UI	confirmGenerationOfSAD
6.3.24	The SSSrv/UI signs the SAD.	SSSrv/UI	signSAD
6.3.25	The SSSrv/UI requests the SAK /OS to add the signature for the SAD to the remote signature process.	SSSrv/UI, SAK/OS	(POST) /digest/remote-signature/{UUID}
6.3.26	The SAK/OS verifies the request to add the signature for the SAD to the remote signing process.	SAK/OS	checkRequestAddSADSignature
6.4 Signing Key Activation			

6.4.1	The SAK/OS requests the SSA to create a signature by submitting a request to the SSA with the signed SADs.	SAK/OS, SSA	requestSigning A remote signature request is sent from the SAK/OS to the SSA. Its format is given by SAP or its specification. The request contains the SAD.
6.4.2	The SSA checks the authorization regarding the request to create a signature.	SSA	checkRequestSigning
6.4.3	The SSA requests the SAM to sign.	SSA, SAM	requestSigning
6.4.4	The SAM checks the request to create a signature.	SAM	checkRequestSigning
6.4.5	The SAM checks whether the Signer is authenticated.	SAM	checkSignerAuthentication
6.4.6	The SAM checks the validity of the declaration of intent to sign.	SAM	checkValidityOfSAD
6.4.7	The SAM requests the KM to send the wrapped key belonging to the KeyID.	SAM, KM	requestWrappedKey
6.4.8	The KM checks the request for delivery of the wrapped key.	KM	checkRequestWrappedKey
6.4.9	The KM responds to the SAM by returning the wrapped key associated with the KeyID.	KM, SAM	returnWrappedKey

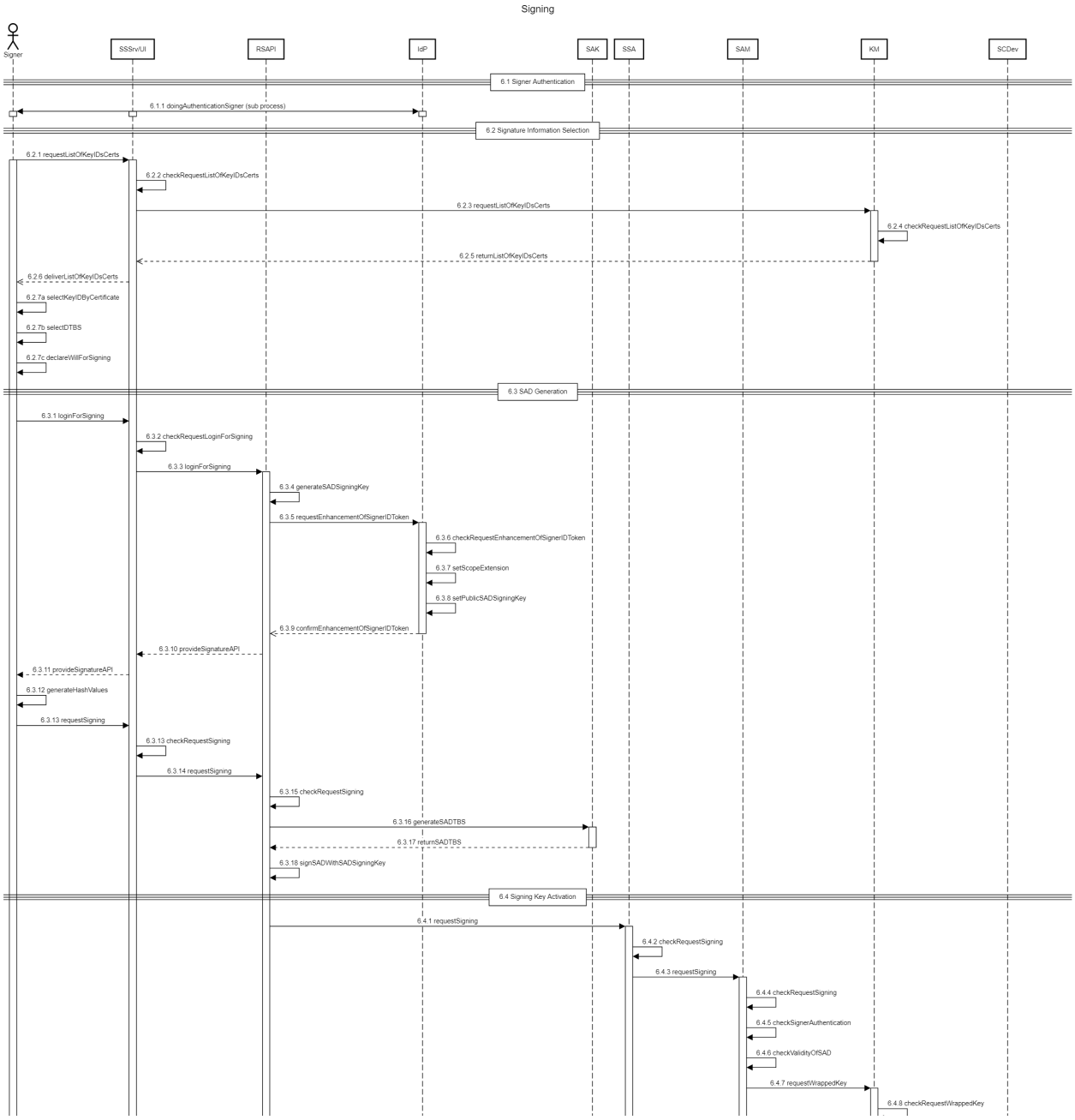
6.4.10	The SAM checks the validity of the signature of the wrapped key.	SAM	checkSignatureOfWrappedKey
6.4.11	The SAM checks whether: the KeyIDs contained in the SAD and the Wrapped match, and whether. the UserID contained in the Wrapped Key and the ID Token of the SAD match.the KeyID between	SAM	checkMatchingOfUserIDKeyID
6.4.12	The SAM verifies that the algorithm chosen by the Signer for signature creation is compatible and acceptable for use.	SAM	checkCompatibilityOfAlgorithm
6.4.13	The SAM activates the private remote signature key.	SAM	activatePrivateKey
6.5 Signature Value Creation			
6.5.1	The SAM requests the SCDev to sign the DTBS/R.	SAM, SCDev	requestSigningOfDTBSR
6.5.2	The SCDev signs the DTBS/R with the signature key referenced by the KeyID.	SCDev	signDTBSR

6.5.3	The SCDev confirms the signature of the DTBS/R by returning the signature value to the SAM.	SCDev, SAM	confirmSigningOfDTBSR
6.5.4	The SAM deactivates the private key.	SAM	deactivatePrivateKey
6.5.5	The SAM responds to the SSA and confirms the creation of the signature value.	SAM, SSA	confirmSigning
6.5.6	The SSA checks the validity of the certificate.	SSA	checkValidityOfCertificate
6.5.7	The SSA responds to the SAK /OS and confirms the creation of the signature value.	SSA, SAK /OS	confirmSigning
6.5.8	The SAK/OS generates the signature container to represent the signed data.	SAK/OS	createSignatureContainer
6.5.9	The SAK/OS checks the signed data and the validity of the Signer certificate.	SAK/OS	checkSignature
6.5.10	The SAK/OS confirms the creation of the (remote) signature to the SSSrv/UI.	SAK/OS, SSSrv/UI	confirmCreationOfSignature

6.5.11	The SSSrv/UI responds to the Signer by handing over the signed data, thus confirming the remote signing.	SSSrv/UI, Signer	confirmSigning

Table 12: Step-by-step description Signing process

5.6.2 Signing using remote SAK



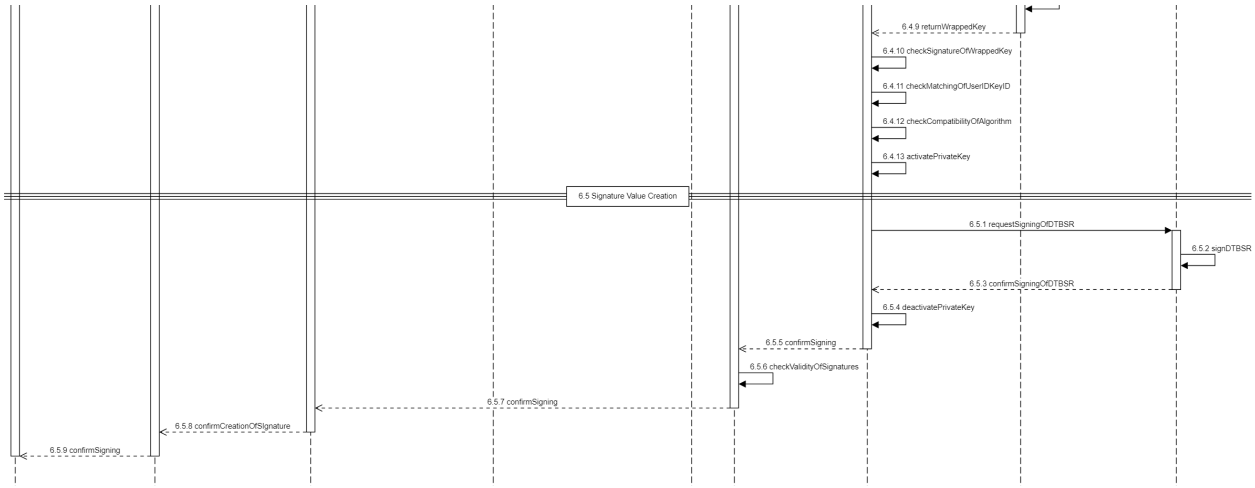


Figure 13: Sequence diagram of the Signing process

Nr.	Step	Components	Description
6 Signing			
6.1 Signer Authentication			
6.1.1	The Signer performs the authentication process.	Signer, SSSrv/UI, IdP	(61 doingAuthentificationSigner) Subprocess that authenticates the Signer based on the identification means used. The result is the ID token for the Signer, which signals that authentication has been successfully performed.
6.2 Signature Information Selection			

6.2.1 (optional)	The Signer requests the list of KeyIDs and associated certificates corresponding to the ID token from the SSSrv/UI.	Signer, SSSrv/UI	requestListOfKeyIDsCerts
6.2.2 (optional)	The SSSrv/UI checks the request for the list of KeyIDs and associated certificates.	SSSrv/UI	checkRequest ListOfKeyIDsCerts
6.2.3 (optional)	The SSSrv/UI queries the KM for the KeyID list that matches the specified ID token.	SSSrv/UI, KM	requestListOfKeyIDsCerts
6.2.4 (optional)	The KM checks the request for the list of KeyIDs.	KM	checkRequest ListOfKeyIDsCerts

6.2.5 (optional)	The KM sends the KeyID list with the corresponding certificate information back to the SSSrv/UI.	KM, SSSrv/UI	returnListOfKeyIDsCerts
6.2.6 (optional)	The SSSrv/UI delivers the identified KeyIDs and associated certificates to the Signer.	SSSrv/UI, Signer	deliverListOfKeyIDsCerts
6.2.7	The Signer selects the certificate and thus the associated KeyID to be used for signing, selects the data to be signed, and makes a declaration of intent to sign.	Signer	selectKeyIDByCertificate selectDTBS declareWillForSigning The declaration of intent is realized via a checkbox. The SHA512 hash of the document and the certificate details of the certificate used for the signature are displayed to the user.
6.3 SAD Generation			
6.3.1	The Signer requests the SSSrv/UI to log in for Signing.	Signer, SSSrv/UI	loginForSigning
6.3.2	The SSSrv/UI checks the request for logging in for Signing.	SSSrv/UI	checkRequestLoginForSigning

6.3.3	The SSSrv/UI requests tu login for Signing at the RSAPI.	SSSrv/UI, RSAPI	loginForSigning
6.3.4	The RSAPI generates the temporary SAD Signing Key.	RSAPI	generateSADSigningKey
6.3.5	The RSAPI requests the enhancement of the Signers ID token at the IdP (handling over the public SAd Signing key).	RSAPI, IdP	requestEnhancementOfSignerIDToken
6.3.6	The IdP checks the request for enhancing theID token of the Signer.	IdP	checkRequestEnhancementOfSignerIDToken
6.3.7	The IdP sets the Signing relevant scope extensionen in the ID token.	IdP	setScopeExtension
6.3.8	The IdP sets the Public SAD Signing Key in thr ID Token.	IdP	setPublicSADSigningKey
6.3.9	The IdP confirms the RSAPI the enhancement of the ID token.	IdP, RSAPI	confirmEnhancementOfSignerIDToken
6.3.10	The RSAPI provides the Signature API to the SSSrv/UI.	RSAPI, SSSrv/UI	provideSignatureAPI

6.3.11	The SSSrv/UI provides the SignatureAPI to the Signer.	SSSrv/UI, Signer	provideSignatureAPI
6.3.12	The Signer generates the Hash Value(s) for the data to be remotely signed.	Signer	generateHashValues
6.3.13	The Signer requests Signing at the SSSrv/UI.	Signer, SSSrv/UI	requestSigning
6.3.14	The SSSrv/UI checks the request for Signing.	SSSrv/UI	checkRequestSigning
6.3.15	The SSSrv/UI requests Signing at the RSAPI.	SSSrv/UI, RSAPI	requestSigning
6.3.16	The RSAPI checks the request for Signing.	RSAPI	checkRequestSigning
6.3.17	The RSAPI requests the SAK for the generation of the SAD.	RSAPI, SAK	requestGenerationOfSAD
6.3.18	The SAK checks the requests for generation of the SAD.	SAK	checkRequestGenerationOfSAD
6.3.19	The SAK generates the SAD.	SAK	generateSAD

6.3.20	The SAK returns the SAD to the RSAPI.	SAK, RSAPI	returnSAD
6.3.21	The RSAPI signs the SAD with the pregenerated SAD Signing Key.	RSAPI	signSADWithSADSigningKey
6.4 Signing Key Activation			
6.4.1	The SAK/OS requests the SSA to create a signature by submitting a request to the SSA with the signed SADs.	SAK/OS, SSA	requestSigning A remote signature request is sent from the SAK/OS to the SSA. Its format is given by SAP or its specification. The request contains the SAD.
6.4.2	The SSA checks the authorization regarding the request to create a signature.	SSA	checkRequestSigning
6.4.3	The SSA requests the SAM to sign.	SSA, SAM	requestSigning
6.4.4	The SAM checks the request to create a signature.	SAM	checkRequestSigning
6.4.5	The SAM checks whether the Signer is authenticated.	SAM	checkSignerAuthentication
6.4.6	The SAM checks the validity of the declaration of intent to sign.	SAM	checkValidityOfSAD

6.4.7	The SAM requests the KM to send the wrapped key belonging to the KeyID.	SAM, KM	requestWrappedKey
6.4.8	The KM checks the request for delivery of the wrapped key.	KM	checkRequestWrappedKey
6.4.9	The KM responds to the SAM by returning the wrapped key associated with the KeyID.	KM, SAM	returnWrappedKey
6.4.10	The SAM checks the validity of the signature of the wrapped key.	SAM	checkSignatureOfWrappedKey
6.4.11	The SAM checks whether: the KeyIDs contained in the SAD and the Wrapped match, and whether. the UserID contained in the Wrapped Key and the ID Token of the SAD match.the KeyID between	SAM	checkMatchingOfUserIDKeyID
6.4.12	The SAM verifies that the algorithm chosen by the Signer for signature creation is compatible and acceptable for use.	SAM	checkCompatibilityOfAlgorithm
6.4.13	The SAM activates the private remote signature key.	SAM	activatePrivateKey
6.5 Signature Value Creation			

6.5.1	The SAM requests the SCDev to sign the DTBS/R.	SAM, SCDev	requestSigningOfDTBSR
6.5.2	The SCDev signs the DTBS/R with the signature key referenced by the KeyID.	SCDev	signDTBSR
6.5.3	The SCDev confirms the signature of the DTBS/R by returning the signature value to the SAM.	SCDev, SAM	confirmSigningOfDTBSR
6.5.4	The SAM deactivates the private key.	SAM	deactivatePrivateKey
6.5.5	The SAM responds to the SSA and confirms the creation of the signature value.	SAM, SSA	confirmSigning
6.5.6	The SSA checks the validity of the certificate.	SSA	checkValidityOfCertificate
6.5.7	The SSA responds to the SAK /OS and confirms the creation of the signature value.	SSA, SAK /OS	confirmSigning
-	The SAK/OS generates the signature container to represent the signed data.	SAK/OS	createSignatureContainer
-	The SAK/OS checks the signed data and the validity of the Signer certificate.	SAK/OS	checkSignature

6.5.8	The SAK/OS confirms the creation of the (remote) signature to the SSSrv/UI.	SAK/OS, SSSrv/UI	confirmCreationOfSignature
6.5.9	The SSSrv/UI responds to the Signer by handing over the signed data, thus confirming the remote signing.	SSSrv/UI, Signer	confirmSigning

Table 13: Step-by-step description Signing process

5.6.3 doingAuthenticationSigner

This process is a sub process of the Signing. It takes place in the Signer authentication part of Signing. It includes the technical authentication processes using different authentication mechanisms. The result of the authentication process is that the identity provider is able to create and provide an ID token for the Signer.

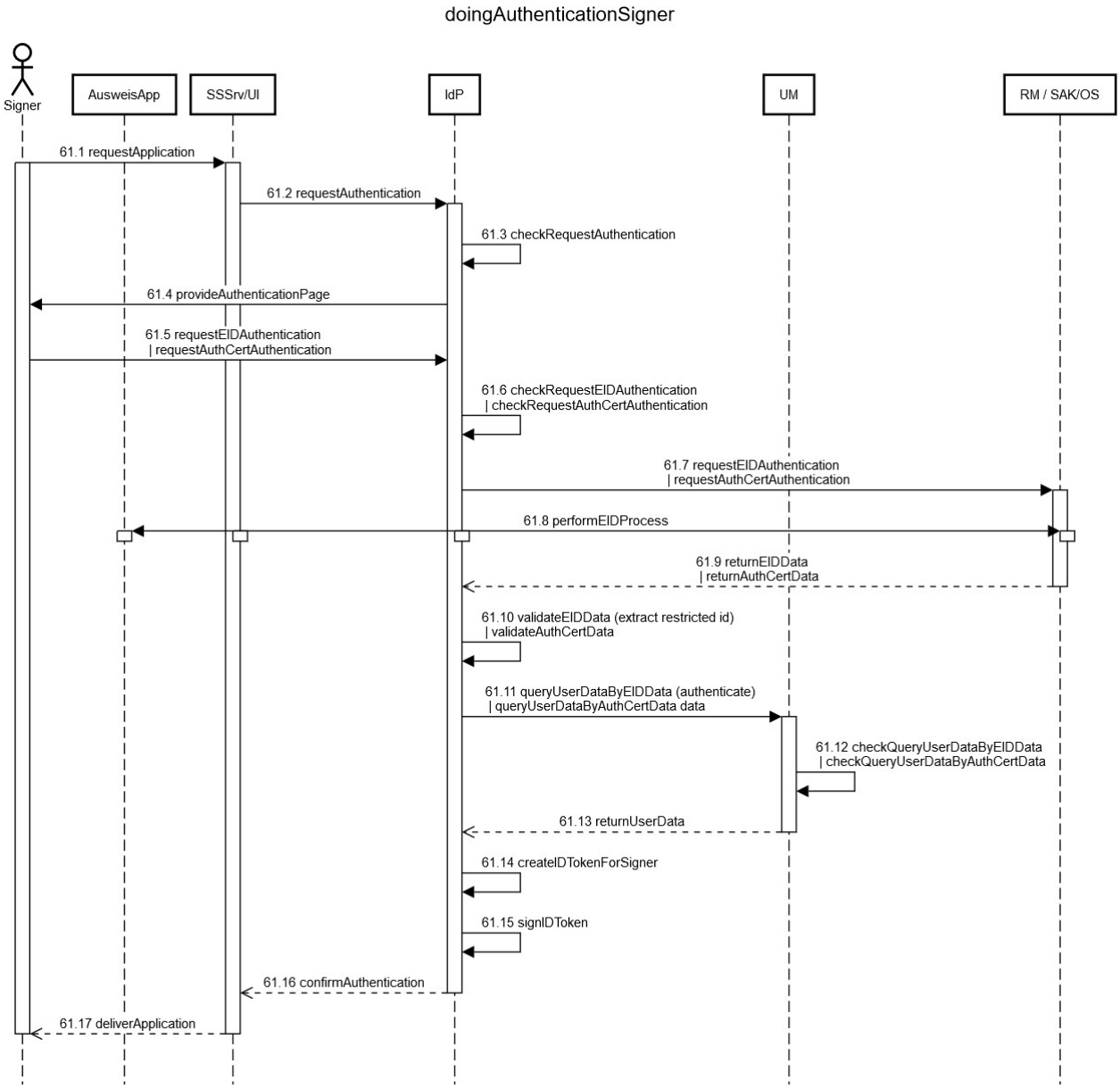


Figure 14: Sequence diagram doingAuthenticationSigner process

Nr.	Step		Components	Description	
	eID Card	Hard Token		eID Card	Hard Token
61 doingAuthenticationSigner					

<table>
<tr><td>61.1</td><td colspan="2">The Signer requests authentication of the Signer at the SSSrv/UI.</td><td>Signer, SSSrv/UI</td><td colspan="2">requestApplication</td></tr>
<tr><td>61.2</td><td colspan="2">The SSSrv/UI requests authentication of the Signer to the IdP.</td><td>SSSrv/UI, IdP</td><td colspan="2">requestAuthentication</td></tr>
<tr><td>61.3</td><td colspan="2">The IdP checks the request for authentication.</td><td>IdP</td><td colspan="2">checkRequestAuthentication</td></tr>
<tr><td>61.4</td><td colspan="2">The IdP provides the Signer with the authentication page.</td><td>IdP, Signer</td><td colspan="2">provideAuthenticationPage</td></tr>
<tr><td rowspan="2">61.5</td><td colspan="2">The undersigned requests at the IdP</td><td rowspan="2">Signer, IdP</td><td rowspan="2">requestEIDAuthentication</td><td rowspan="2">requestAuthCertAuthentication</td></tr>
<tr><td>a eID based authentication.</td><td>a certificate based authentication.</td></tr>
<tr><td>61.6</td><td colspan="2">The IdP checks the request for authentication.</td><td>IdP</td><td>checkRequestEIDAuthentication</td><td>checkRequestAuthCertAuthentication</td></tr>
<tr><td rowspan="2">61.7</td><td colspan="2">The IdP demands</td><td rowspan="2">IdP, RM/SAK</td><td rowspan="2">requestEIDAuthentication</td><td rowspan="2">requestAuthCertAuthentication</td></tr>
<tr><td>a eID based authentication for the RM.</td><td>a certificate based authentication for the IdP's own SAK
/OS.</td></tr>
</table>

61.8	RM, IdP, SSSrv /UI and IDApp perform the eID procedure.	The IdP's own SAK /OS reads the data from the authentication certificate.	RM (IdP, SSSrv/ UI, Ausweis App) / SAK/ OS	performEIdProcess	readCertificateData
61.9	The IdP is supplied		RM / SAK /OS, IdP	returnEIDData	returnAuthCertData
	with the eID data by the RM.	with the data of the authentication certificate by the IdP's own SAK/OS.			
61.10	The IdP validates the		IdP	validateEIDData	validateAuthCertData
	eID data (extracts the Restricted ID).	data of the authentication certificate.			
61.11	The IdP queries user data at the UM		IdP, UM	queryUserData ByEIDData	queryUserData By AuthCertData
	according to the read RestrictedID.	according to the read authentication certificate.			
61.12	The UM checks the request to query user data based on the		UM	checkQuery User DataByEIDData	checkQueryUserData ByAuthCertData

	eID data.	data of the authentication certificate			
61.13	The UM sends the user data back to the IdP.		UM, IdP	returnUserData	
61.14	The ID token is generated according to the authenticated Signer and the queried user data.		IdP	createIDTokenForSigner	
61.15	The IdP signs the ID token.		IdP	signIDToken	
61.16	The IdP confirms the authentication by returning the signed ID token to the SSSrv/UI.		IdP, SSSrv /UI	confirmAuthentication	
61.17	The SSSrv/UI delivers the application to the Signer.	SSSrv/UI, Signer	deliverApplication		

Table 14: Step-by-step description doingAuthenticationSigner process

5.7 Server Signing Processes: SAM Maintenance

The Administrators are allowed to execute the following management operations on the SAM subsystem per manageSAM.sh or manageFW.sh script:

Operation	Command

Start	manageSAM [-v] start <instanceId>
Stop	manageSAM [-v] stop <instanceId>
Check Code Integrity	manageSAM [-v] integrity code <instanceId>
Check Data Integrity	manageSAM [-v] integrity data
Initialize SAM Firmware	manageFW [-v] init LogonPass=<user>,<credential> <MBK slot id>
Set IdP Public Keys	manageFW [-v] setidpkeys LogonPass<user>,<credential> <certfile>

Table 15: SAM Maintenance operations

The operation Check Code Integrity is carried out for the subsystem of the SAM Service module and for the SAM MAN module by the Linux command sha512sum. The expected checksums are stored in a configuration file. The expected checksums are stored in a configuration file, which must be created in advance using the shell script checksumSAM.sh.

The operation Check Data Integrity is performed by calling the appropriate methods via the REST Service provided by the module SAM Service.

The Operation Set IdP Public Keys imports a certain number of public keys to the SAM firmware (when it is started) to verify the ID tokens issued and signed by the IdP. 'Initialize SAM Firmware' initializes the SAM Firmware and for example derives key material from the used MBK necessary for providing the functions of the SAM Firmware.

The operation TOE_Maintenance is performed by adjust the SAM Service related XML configuration file as it is described in chapter 5.2.3.1 of [AGD_PRE].

6 Server Signing Components

In the following, the components relevant for server signatures are presented, whereby these are listed by means of a further development of the general server signing scenario embedded in a common CA environment. The connection between the communication relationships of the various components is also taken into account.

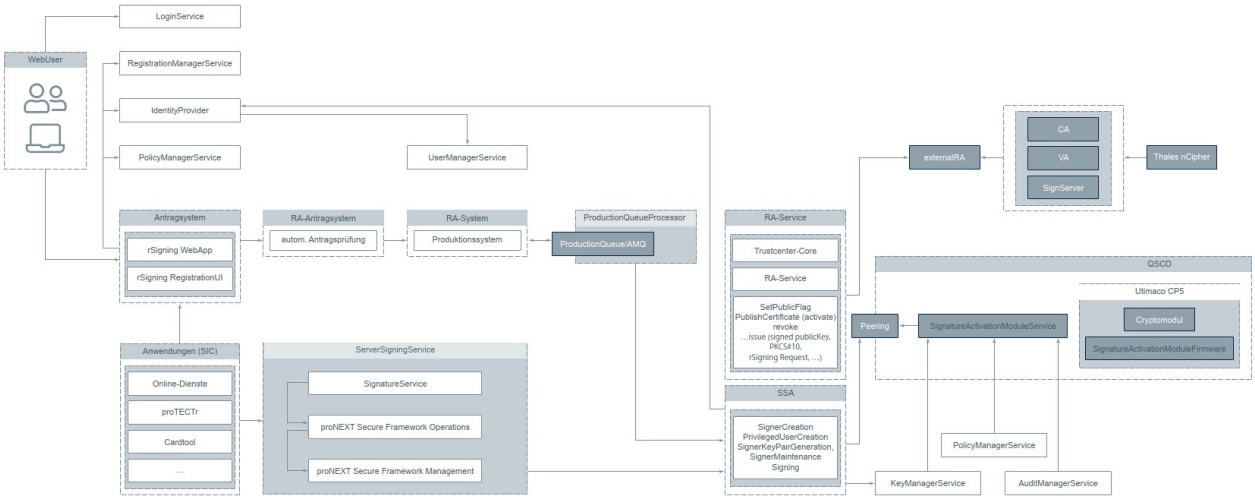


Figure 15: Embedding of the server signing components into a CA environment

6.1 QSCD

6.1.1 SignatureActivationModule

The Signature Activation Module (SAM) is a control unit for the cryptographic module. In particular, it ensures that only the owner of a key can access it and thus sign it. It consists of two modules: the SAM service and the SAM firmware. The SAM firmware is integrated into the cryptographic module and its specially secured environment.

6.1.2 SignatureActivationModuleService

The SignatureActivationModuleService is an essential part of the Signature Activation Module (SAM). The other part is a firmware module that is installed in the Utimaco CP5 HSM.

The task of the SignatureActivationModuleService is to generate and manage the remote signature keys and to ensure authorized access to them. Signatures using remote signature keys can only be generated by the SAM.

There are dependencies to the Server Signing Application (SSA), the SAM firmware, the AuditManagerService and the KeyManager. In particular, REST(HTTPS) and WebSocket/Peering based interfaces are used. In contrast to the CP5, the CXI Java based interface is used in particular.

The interfaces of the SAMS are secured by different security features. The websocket connection should be secured with TLS client auth and certificate pinning. The Create_NewSigner_interface can be invoked with a signature of the message by a Privileged User Technical (RSA/ECDSA + SHA-256). All interfaces can be called by specifying an identity token.

6.1.3 SignatureActivationModuleFirmware

The SignatureActivationModuleFirmware is an essential part of the Signature Activation Module (SAM). The other part is the service module, which is used outside the CP5.

The task of the SignatureActivationModuleFirmware is to create and manage remote signature keys and to ensure authorized access to them. Signatures using remote signature keys can only be generated by the SAM.

6.1.4 Utimaco CP5 HSM

The Utimaco CP5 HSM is a Common Criteria certified hardware security module based on the CryptoServer Se Gen2 hardware platform, which fulfills the requirements of the eIDAS Protection Profile (PP) [EN419 221-5] as well as other various technical ETSI standards (in particular [EN319401] and [EN319411]). This makes it particularly suitable for eIDAS compliant qualified signature creation and remote signing. Other areas of application include the issuing of (qualified) certificates, OCSP responses, and time stamps.

In the context of remote signature applications, a so called Signature Activation Module (SAM) is required to authorize signatures. A cryptographic module certified according to [EN419221-5], such as the Utimaco CP5 HSM, and a signature activation module in the form of an "internal SAM" that meets the requirements of [EN419241-2] together form a QSCD.

For the installation of the Utimaco CP5 HSM the installation manual [InstCP5] has to be consulted. For the installation of the SAM firmware the installation guide [AGD_PRE] has to be consulted.

6.2 proNEXT Key Manager

The Key Manager Service manages cryptographically relevant objects of various forms on the basis of the Key Management Interoperability Protocol [KMIPv20]. A distinction is made between base objects and managed objects. Base objects are information that specifies a managed object. Base Objects include, for example, Attributes, Key Value and Key Wrapping Data.

A Managed Object is an object with cryptographic content that is managed by the Key Lifecycle Management System (KLMS). This includes the various keys and certificates. Templates can also be created to allow the administrator of a KLMS to group together attributes of frequently used processes. For example, a template can be created for a symmetric key in which the algorithm and length of the key are defined. When a key is to be created according to these specifications, the name of the template is passed instead of the desired attributes. To store other objects to be kept secret, the Secret Data Object (e.g. for passwords) or the Opaque Object are used. The data in the Opaque Object does not have to be interpretable by the server. For example, a key is stored even though the server does not support the encryption algorithm used.

The Key Manager can be used as a central key management and thus increase the general data security.

Access to the Key Manager's interfaces is token based. For this purpose, a signed token is created by the IdP, which contains the corresponding user information that can be used to check access authorization to the Key Manager interfaces.

6.3 proNEXT Policy Manager

The Policy Manager is a service in the field of secure identity and access management. In this context, the Policy Manager delivers policies signed in electronic form, which represent user access authorizations related to requested use cases, to business applications.

The Policy Manager works with the concept of roles, which correspond to a sum of authorizations or access rules. Roles are activity related, permissions can be marked as inheritable / assignable. In order to access a specific resource, a suitable role must be designated for it.

A business application can query the above mentioned policies for access and authorization checks via the interfaces of the Policy Manager. Access to the Policy Manager interfaces is token based. For this purpose, a signed token is created by the IdP, which contains the corresponding user information that can be used to check access authorization to the Policy Manager interfaces.

6.4 proNEXT Audit Manager

The AuditManager (AM) is a service in the area of secure identity and access management. It is a service and provides an audit function for various specialized applications.

The AM is based on the implementation of the CeSECore core. Authorization checks are performed using ID tokens and X.509 certificates. Information for creating an audit entry can be supplied by the specialist applications via REST interface.

The Audit Manager has a number of administrative functions. These include in particular the creation of module specific storage areas, the assignment of authorizations via SSL client authentication or ID tokens, the assignment of log backends to storage areas, the management of additional log providers, integrity protection both in the and the configuration. As well as search, verification and export of audit logs.

Operations are provided using HTTP GET/POST methods. Access to the Audit Manager interfaces is token based. For this purpose, a signed token is created by the IdP, which contains the corresponding user information that can be used to check access authorization to the Policy Manager interfaces. The required peripheral systems are a CeSECore core, a DBMS, an AuditManagerProxy and a PolicyManagerService.

6.5 proNEXT Server Signing Application

The Server Signing Application (SSA) is a software that acts as a kind of proxy. It interacts directly with the SAM and uses a cryptographic module to generate, hold, and use the signing keys. The SSA provides an interface to the SAM of the cryptographic module. All requests to the SAM (e.g., regarding signatures to be generated) by the SAK/OS or users of the SAM shall be received by the SSA and forwarded accordingly. The SSA requires each Signer to successfully identify and authenticate themselves before allowing any actions that may affect the SAM. The SSA is responsible for screening requests and managing audit logs. It may maintain Signer authentication for a specified period of time and/or for a specified number of signatures. The SSA optionally communicates with the SAM to provide data relevant to their function. Used to provide a registration service in accordance with [EN319411-1].

The SSA is called in the production process by the ProductionQueueProcessor. Surrounding applications and services are in particular the SignatureActivationModuleService, the proNEXT IdP, the SecureFramework Operations, and Audit Manager and Key Manager. The interfaces used are REST(HTTPS) and WebSocket (peering) based.

The interfaces of the SSA are secured via different security features. If the SSA is called in the production process from the ProductionQueueProcessor, the production request must include a

data field signed by the RegistrationManager that contains the trusted eID data. The signing REST interface expects a valid ID token in the SAD. Other REST interfaces can only be invoked with a valid ID token in the Authorization header.

Depending on the interface, the ID token must include one of two roles: 'RemoteSignature-Signer' or 'RemoteSignature-PrivilegedUser'.

6.6 Server Signing Service

6.6.1 proNEXT Secure Framework

The proNEXT Secure Framework is a signature application component for creating and verifying electronic signatures. In particular, the proNEXT Secure Framework provides interfaces for applying a remote signature. Documents in various electronic formats - including those in PDF format - can be processed in single or batch mode. The following functions are provided:

Creation of advanced or qualified electronic signatures over single or multiple documents (a secure signature creation device is required to create a qualified electronic signature.

Validation of electronic signatures of single or multiple documents Verification of documents to be signed for active or hidden content

Further security function is:

Integrity protection: to protect against unnoticed modifications, the integrity of the application is checked every time it is started or when required.

The application provides an application programming interface (API) that allows other applications to call the above functions through this API.

The proNEXT Secure Framework is a Java based client / server application. The client component (proNEXT Secure Framework Operations) can be used under Microsoft Windows, Apple Mac OS X or Linux operating system. For the server component (proNEXT Secure Framework Management) a Linux operating system is required. Furthermore, a card terminal with secure PIN entry can be used.

The client component is used to generate hash values, verify documents, communicate with smart cards and provide interfaces for applying server signatures. On the server side, certificate information is collected and verification reports are generated.

An integrity protection mechanism enables consistency checking of the proNEXT Secure Framework installation.

6.6.2 Signature Service

The SignatureService can be used to generate various signatures. Currently, these are the deputy signature and the remote signature.

The signature application component proNEXT Secure Framework is used to create signatures. The signature containers generated by this component comply with the proNEXT Secure Framework specification PKCS7 [RFC2315] or PDF inline [ISO32000-1].

There are dependencies to the following surrounding applications and services: proNEXT Secure Framework Operations, AuditManagerService.

Access to the SignatureService interfaces is token based. For this purpose, a signed token is created by the IdP, which contains the corresponding user information that can be used to check the access authorization to the interfaces of the SignatureService. The signature of the transferred token is checked against the configured IdP certificate.

6.7 Remote Signature API

The Remote SIgnature API is a component that is installed in the signer's environment and can be accessed from a browser or a mobile device, for example. This component participates in the signature activation protocol (SAP) and generates the SAD. Provides the link between the signer and the signature process (linking the document to be signed, the remote signature key used by the signer to sign, and the data that authenticates the signer). Communicates with the SSA for the purpose of transferring the generated SAD to the SAM. Can be used alternatively to the SAK/OS.

6.8 Registration Manager

The Registration Manager is an infrastructure component for creating and managing user accounts. In case of lost login data, passwords can be generated and sent by email.

When requesting the user name for an email address, the user will directly receive an email with its user name. When requesting a new login password for an email address, the user receives an email with a confirmation link (with limited validity), after which a new login password is generated and sent to the user by email.

ID card data can be read out via the ident interface. This data can then be used by the calling application to create/update a user account or to authenticate a user.

There are dependencies on the following applications and services: proNEXT IdP, User Manager, for EID an eID client comparable to the AusweisApp2.

The Registration Manager must be secured via TLS. This TLS certificate is 'interleaved' with the eID authorization certificate, which means: if the TLS certificate is replaced, the eID authorization certificate must also be reapplied for or updated at the eID server operator (Governikus).

In addition to TLS, communication between the Registration Manager, the AusweisApp2 and the autent server is secured by encryption and signature of the content data (SAML protocol). This ensures not only data integrity but also the authorization of the services.

The data read from the ID card is signed and stored in encrypted form in a . Only the service that initiated the readout process is then able to decrypt the data after retrieving it from the Registration Manager. If, in a second step, a new user data record is to be created in the User Manager with this read out ident data or an existing data record is to be supplemented with it, the decrypted content signed by the Registration Manager must in turn be transferred to the Registration Manager. This ensures that the data actually originates from an ID card and is unchanged.

Some REST interfaces of the Registration Manager require a valid ID token issued by the IdP for authentication.

6.9 User Manager

The User Manager is an application for managing users, groups, applications, roles, certificates, group and role assignments.

Required peripheral systems are: OpenLDAP, proNEXT IdP, Registration Manager, AuditManagerService. Interfaces called from the User Manager are: HTTPS(REST) (IdP, RM, AMS) and LDAP(StartTLS) (LDAP).

The creation of the central infrastructure component for the management of users, user roles and organizational structures avoids the duplication of such information in specialized systems and consequently reduces the administrative effort required. In addition, the UM concept provides precise specifications for its internal authorization structure and thus allows fine granular administration which can be defined for different user groups. The known user / role data structure is extended by a new group entity, which enables the logical combination of users and allows the definition of administration areas through a hierarchical structure.

Furthermore, groups allow to give a user different roles in the context of his current group membership. Asymmetric key material (in the form of public certificates and private keys) can be stored with users as well as in groups.

In addition to the ability to authenticate directly (active and passive authentication), the UM concept handles the secure passing of authentication information between background services without requiring user interaction again. This eliminates the need for technical "superusers" which often pose a significant security risk due to their liberal legal model.

All read and write access is controlled via central components. Direct access to the attribute store (in this concept an LDAP system) is technically not possible. This is to prevent (erroneous) third party accesses from posing a threat to the data integrity of the user administration.

6.10 proNEXT Identity Provider

The proNEXT identity provider 3 is a further development of the proNEXT IdP based on Keycloak. The extensive functionality of the previous IdP will only be gradually incorporated into Keycloak. In the first version only the login with user name and password as well as with the german identity card is supported. Keycloak itself offers a very extensive documentation, which is referred to here again and again.

The proNEXT identity provider (proNEXT IdP) is an authentication component with single sign on technology. By means of the proNEXT IdP it is possible to provide a central authentication possibility which can be accessed from different specialized applications. By default, the proNEXT IdP offers a login via user name and password. However, this functionality can be extended so that, for example, the new german ID card or SSL client certificates can also be used. The proNEXT IdP is multi client capable, so that specific authentication options can be configured for each client.

Both the proNEXT IdP and the service providers accessing it must be time synchronized via an NTP server. This should usually be a public time server. If all components have access to a local NTP service, a local server can also be used.

6.11 Login Service

The LoginService is a service provider that is used as a communication interface between the IdP and an application. This communication interface is used in the LSP (Local Service Provider) context to enable token based login to the NPM repository via the identity proxy.

The identity proxy works as a reverse proxy based on the HTTP/HTTPS protocol, depending on the configuration of the application server on which the identity proxy is running. This authentication interface is used in the Local Service Provider (LSP) context to enable token based logon to the NPM repository.

After the LSP logs on to the identity proxy, the identity token passed, issued by the proNEXT IdP for the LSP, is checked. If the check is successful, the request is forwarded to the NPM repository and the requested artifact is returned to the requestor. If an error occurs during the identity token check, the request is rejected with an error message.

The identity token is transmitted to the identity proxy as an authorization bearer header in the HTTP/HTTPS request.

Communication with the NPM repository is also done via HTTP/HTTPS, but authorization basic is used here instead of authorization bearer.

The identity proxy software component consists of a Java EE Enterprise application that provides an HTTP

/HTTPS interface for communication with the LSP. The identity proxy software component itself invokes the HTTP/HTTPS interface of an NPM repository to query software artifacts.

The identity proxy is an authentication interface for an NPM repository (e.g., Artifactory). This authentication interface is used in the Local Service Provider (LSP) context to enable token based logon to the NPM repository. For token based login, the LSP requests a signed identity token from the proNEXT IdP. This identity token is returned by proNEXT IdP only if the authorization at proNEXT IdP was successful. The signed identity token is passed to the identity proxy in the HTTP/HTTPS header during the request. The identity proxy checks the validity of the identity token (signature, expiration time) and then forwards the request to the NPM repository.

The LoginService enables the connection of an application to the IdP infrastructure.

7 Inferface Specifications

7.1 SAM Peering Interface

The SAM Peer Interface is an interface that manages requests from remote systems. The SAM initiates connections to a peer connector via the SAM Peer Interface.

A peer connector is a representation of a remote instance. More precisely, these are instances that are known to the system currently under consideration and to which it can establish a connection. Peer Connectors are characterized in particular by a connection URL and TLS certificate information, as well as a pool of reusable outgoing connections. The identity of an instance consists of a client SSL X509 certificate and a key pair in one of the instance's crypto tokens.

Connections are made over mutually authenticated HTTPS and are also designated as a channel. The first connection to a peer using the same client certificate undergoes a full authentication check.

To allow connections, a new peer connector must be added to the system. The connection must be checked for mutual trust. Messages are authorized. Furthermore, roles and access rules must be assigned to the peer connector. The following configurations must be set up to obtain the necessary roles and thus correct access rules:

Role	Allowed SAM PI Operations
Privileged User	Create_New_Privileged_User, Create_New_Signer, Signer_Maintenance, Generate_Signer_Key_Pair, Delete_Signer_Key_Pair
Signer	Signer_Maintenance, Generate_Signer_Key_Pair, Delete_Signer_Key_Pair, Signing

Privileged User Technical	Create_New_Signer

Table 16: Roles and allowed operations of the SAM Peer Interface

7.1.1 SAM Peering Interface: Create New Privileged User

The following figure shows an overview of the process of creating a new Privileged User by the SAM service.

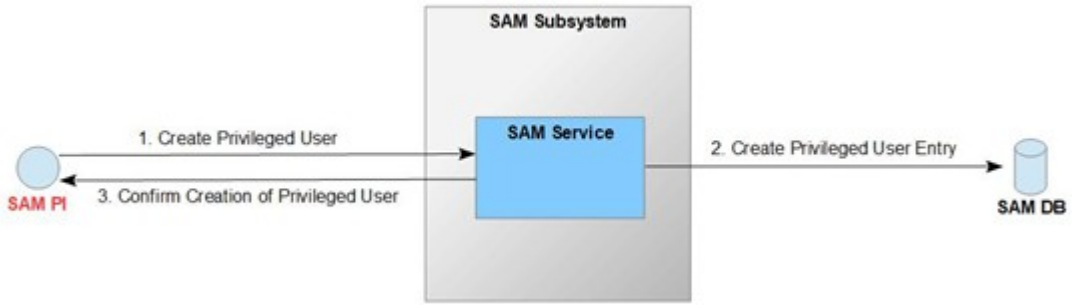


Figure 16: Overview of the Create_New_Privleged_User operation flow

When retrieving the request, the SAM service checks if the requester is authenticated and identified (based on the processes and configuration presented earlier).

If the authenticity and identity check is successful, the SAM service creates a new Privileged User based on the data it received with the request. More precisely, the SAM service stores the given user information in the form of a user entry in the SAM DB. Otherwise, an error is returned.

A user entry has the following structure: {UserID, Role, Certificates}. The 'UserID' is a UUID. For a privileged user, 'Role' has the value 'PrivilegedUser'. 'Certificates' can contain one or more certificates.

Purpose	This operation is invoked by the requestor to create a new privileged user.
Interface	SAM Peer Interface (see [IF_SPI])
Method of use	The operation is called by client software according to the SAM peer interface specification (see [IF_SPI]).

Inputs	Parameter	Type	Description
	userid	String	UserID of the user to be created
Output	Parameter	Type	Description
		void	A result without error indicates a positive operation
Error Messages	Type		Reason
	ContainerParseError		If the input cannot be parsed into a message container
	MessageParseError		If the input data cannot be parsed into the specified message type
	UnknownMessageType		The contained message type was not known to the processor
	UnknownCredentialType		The credential type included was not known to the processor
	AuthenticationError		The credential could not be validated
	UnknownUser		The credential could not be associated with a user
	NotAuthorized		If the acting user is not authorized to perform the current operation

	NotOperational	A component (e.g. , HSM) is not available or not in an operational state	
	UnexpectedError	When an error condition occurs that was not anticipated	
	InvalidInput	When the set of certificates is empty	
Security Audit Log Entry	In case of success An audit record is created that contains at least the following information:		
	▪ the acting UserID ▪ the timestamp of the action ▪ the name of the action ▪ that the action was successful ▪ the UserID of the new Privileged User		
	In case of an error		

	The following errors emit regular log messages, but no audit log entries (since no acting user could be inferred):	
	▪ ContainerParseError ▪ UnknownCredentialType ▪ AuthenticationError ▪ UnknownUser	
	Otherwise	
	An audit record is created containing at least the following information:	
	▪ the acting UserID ▪ the timestamp of the action ▪ the name of the action ▪ that the action failed ▪ a brief description of the cause of the failure.	

Table 17: Description of the Create_New_Privileged_User operation

7.1.2 SAM Peering Interface: Create New Signer

The following figure shows an overview of the process of creating a new Signer by the SAM service.

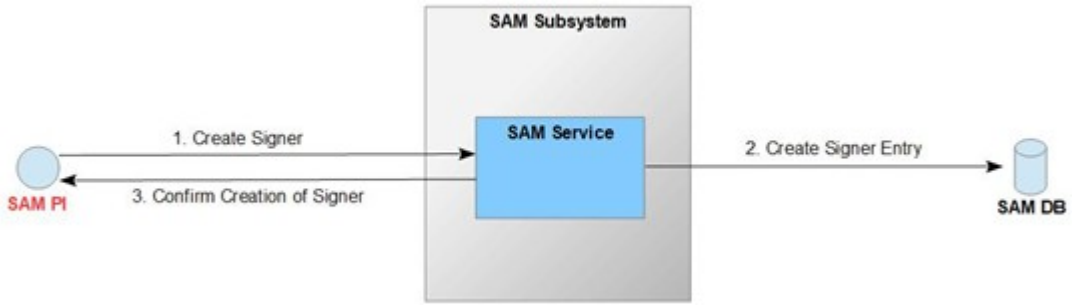


Figure 17: Overview of the Create_New_Signer operation flow

A Privileged User or Privileged User Technical uses the SAM peer interface to request the operation. Authentication and identification of the requesting user is required beforehand.

When the request is retrieved, the SAM service verifies that the requestor is authenticated and identified (based on the processes and configuration presented earlier).

If the authenticity and identity check is successful, the SAM service creates a new Signer based on the data it received with the request. More precisely, the SAM service stores the given user information in the form of a user entry in the SAM DB. Otherwise, an error is returned.

A user entry has the following structure: {UserID, Role, Certificates}. The 'UserID' is a UUID. For a Signer, 'Role' has the value 'Signer'. Certificates' can contain one or more certificates.

Purpose	This operation is invoked by the requestor to create a new Signer.		
Interface	SAM Peer Interface (see [IF_SPI])		
Method of use	The operation is called by client software according to the SAM peer interface specification (see [IF_SPI]).		
Inputs	**Parameter**	**Type**	**Description**
	userid	String	UserID of the user to be created

	certificates	Binary[]	Certificates of the IdP authorized to sign Identity Tokens for this user (multiple certificates are allowed to enable certificate rollover).
Output	**Parameter**	**Type**	**Description**
		void	A result without errors indicates a positive operation.
Error Messages	**Type**		**Reason**
	ContainerParseError		If the input cannot be parsed into a message container
	MessageParseError		If the input data cannot be parsed into the specified message type
	UnknownMessageType		The contained message type was not known to the processor
	UnknownCredentialType		The credential type included was not known to the processor
	AuthenticationError		The credential could not be validated
	UnknownUser		The credential could not be associated with a user

	NotAuthorized	If the acting user is not authorized to perform the current operation
	NotOperational	A component (e.g. , HSM) is not available or not in an operational state
	Unexpected Error	When an error condition occurs that was not anticipated
	InvalidInput	When the set of certificates is empty

Security Audit Log Entry	In case of success An audit record is created that contains at least the following information
	▪ the acting UserID ▪ the timestamp of the action ▪ the name of the action ▪ that the action was successful ▪ the UserID of the new Signer
	In case of an error
	The following errors emit regular log messages, but no audit log entries (since no acting user could be deduced):

	▪ ContainerParseError ▪ UnknownCredentialType ▪ AuthenticationError ▪ UnknownUser
	Otherwise
	An audit record is created containing at least the following information
	▪ the acting UserID ▪ the timestamp of the action ▪ the name of the action ▪ that the action failed ▪ a brief description of the cause of the failure

Table 18: Description of the Create_New_Signer operation

7.1.3 SAM Peering Interface: Signer Maintenance

The following figure shows an overview of the process of managing a Signer through the SAM service.

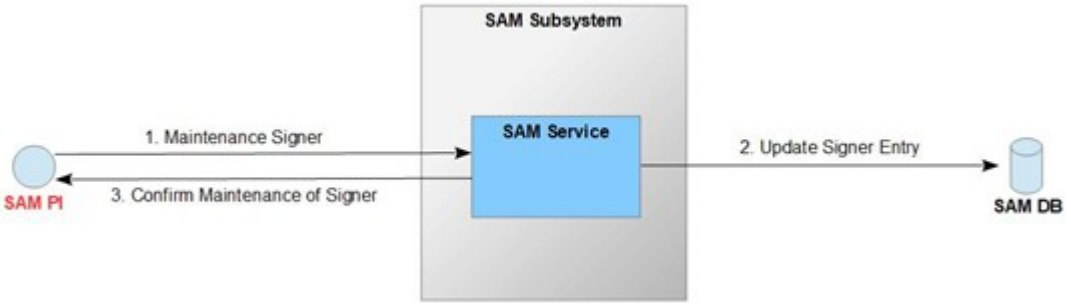


Figure 18: Overview of the Signer_Maintenance operation flow

A Privileged User or Signer uses the SAM peer interface to request the operation. Authentication and identification of the requesting user is required beforehand.

When the request is retrieved, the SAM service verifies that the requestor is authenticated and identified (based on the processes and configuration shown above).

If the authenticity and identity check is successful, the SAM service maintains a Signer's user record based on the data it received with the request. More specifically, the SAM service updates a Signer's authentication data in the form of assigned certificates in the SAM DB. Otherwise, an error is returned.

Purpose	This operation is invoked by the requestor to update a user's credential validation data (the IdP certificate).		
Interface	SAM Peer Interface (see [IF_SPI])		
Method of use	The operation is called by client software according to the SAM peer interface specification (see [IF_SPI]).		
Inputs	**Parameter**	**Typ**	**Description**
	userid	String	UserID of the user to be changed
	certificates	Binary[]	Certificates of the IdP authorized to sign Identity Tokens for this user (multiple certificates are allowed to enable certificate rollover)
Output	**Parameter**	**Typ**	**Description**
		void	A result without errors indicates a positive operation.

Error Messages	Typ	Reason
	ContainerParseError	If the input cannot be parsed into a message container
	MessageParseError	If the input data cannot be parsed into the specified message type
	UnknownMessageType	The contained message type was not known to the processor
	UnknownCredentialType	The credential type included was not known to the processor
	AuthenticationError	The credential could not be validated
	UnknownUser	The credential could not be associated with a user
	NotAuthorized	If the acting user is not authorized to perform the current operation When a regular user specifies the UserID of another user
	NotOperational	A component (e.g. , HSM) is not available or not in an operational state
	UnexpectedError	When an error condition occurs that was not anticipated

	InvalidInput	When the set of certificates is empty If the certificate set does not contain the currently used certificate in case of self modification

Security Audit Log Entry	In case of success An audit record is created that contains at least the following information
	▪ the acting UserID ▪ the timestamp of the action ▪ the name of the action ▪ that the action was successful ▪ the UserID of the changed user
	In case of an error
	The following errors output regular log messages, but no audit log entries (since no acting user could be inferred):
	▪ ContainerParseError ▪ UnknownCredentialType ▪ AuthenticationError ▪ UnknownUser
	Otherwise
	An audit record is created containing at least the following information

	▪ the acting UserID ▪ the timestamp of the action ▪ the name of the action ▪ that the action failed ▪ a brief description of the cause of the failure

Table 19: Description of the Signer_Maintenance operation

7.1.4 SAM Peering Interface: Generate Signer Key Pair

The following figure shows an overview of the process of generating a key pair for a Signer by the SAM service.

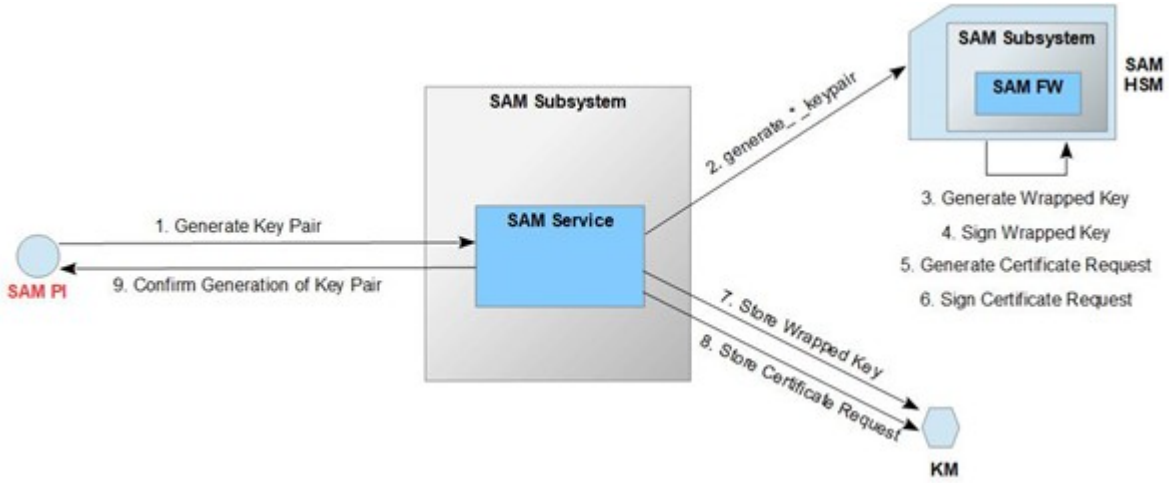


Figure 19: Overview of the Generate_Signer_Key_Pair operation flow

A Privileged User or Signer uses the SAM Peer Interface (SAM PI) to request the operation. Authentication and identification of the requesting user is required beforehand.

When the request is retrieved, the SAM service verifies that the requestor is authenticated and identified (based on the processes and configuration outlined above).

If the authenticity and identity check is successful, the SAM service generates the Signer's key pair. Specifically, the SAM service requests

Requests the cryptographic module (SAM HSM) to generate a key pair,

The SAM firmare

generates the Wrapped Key based on the generated key pair, signs the Wrapped Key,

generates a certificate request based on the generated key pair, signs the certificate request

sends both the Wrapped Key and the certificate request back to the SAM service,

The SAM service then requests storage of the Wrapped Key in the Key Manager (KM), stores the signed certificate request in the key manager (KM). Otherwise, an error is returned.

Keys generated by the CM are exported as a secured key using the CM's module key (CM wrapped key) and stored as part of a signed container (Wrapped Key) in the Key Manager (KM). A key stored outside the generating CM is protected in terms of confidentiality and integrity. The integrity of keys is protected by using the wrapped key structure, which is signed before it is stored in the KM. Key confidentiality is protected by using the CM's module key for encryption, resulting in the CM Wrapped Key.

A Wrapped Key has the following structure: {CMwrappedKey, KeyID, Metadata}. Metadata' contains metadata specific to the remote signature, such as the ID of the associated Signer.

Purpose	This process is invoked by the requestor to generate a new key pair and assign it to a specific user.		
Interface	SAM Peer Interface (see [IF_SPI])		
Method of use	The operation is called by client software according to the SAM peer interface specification (see [IF_SPI]).		
Inputs	**Parameter**	**Typ**	**Description**
	userid	String	UserID of the user for whom the key pair is to be generated
	algorithm	String	Algorithm (EC or RSA)

	parameters	RSAparameters | ECparameters	Algorithm specific parameters: RSAparameters: key length in bits. ECparameters: named curve as OID
Output	**Parameter**	**Typ**	**Description**
	keyid	String	The KeyID of the generated key pair
	publickey	Binary	A public key signed with a private key used exclusively by the SAM
Error Messages	**Typ**		**Reason**
	ContainerParseError		If the input cannot be parsed into a message container
	MessageParseError		If the input data cannot be parsed into the specified message type
	UnknownMessageType		The contained message type was not known to the processor
	UnknownCredentialType		The credential type included was not known to the processor

	AuthenticationError	The credential could not be validated

	UnknownUser	The credential could not be associated with a user
	NotAuthorized	If the acting user is not authorized to perform the current operation When a regular user specifies the UserID of another user
	NotOperational	A component (e.g. , HSM) is not available or not in an operational state
	UnexpectedError	When an error condition occurs that was not anticipated

Security Audit Log Entry	In case of success An audit record is created that contains at least the following information ▪ the acting UserID ▪ the timestamp of the action ▪ the name of the action ▪ that the action was successful ▪ the UserID for which the key was generated ▪ the KeyID of the key pair that was generated In case of an error The following errors emit regular log messages, but no audit log entries (since no acting user could be inferred): ▪ ContainerParseError ▪ UnknownCredentialType ▪ AuthenticationError ▪ UnknownUser Otherwise An audit record is created containing at least the following information ▪ the acting UserID ▪ the timestamp of the action ▪ the name of the action ▪ that the action failed ▪ a brief description of the cause of the failure

Table 20: Description of the Create_Signer_Key_Pair operation

7.1.5 SAM Peering Interface: Delete Signer Key Pair

The following figure shows an overview of the process of deleting a key pair of a Signer by the SAM service.

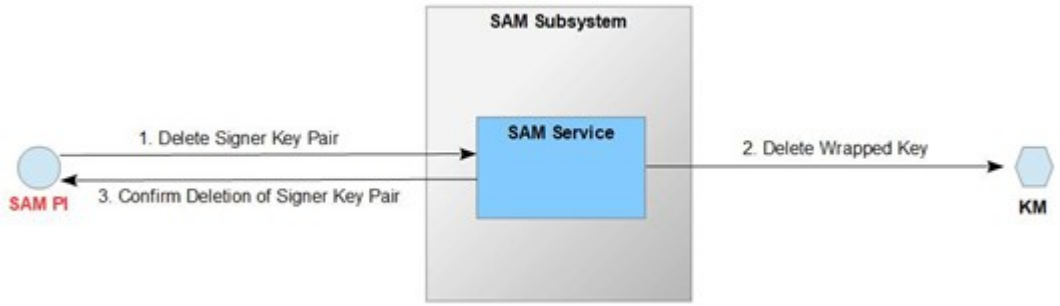


Figure 20: Overview of the Delete_Signer_Key_Pair operation flow

A Privileged User or Signer uses the SAM peer interface to request the operation. Authentication and identification of the requesting user is required beforehand.

When the request is retrieved, the SAM service verifies that the requestor is authenticated and identified (based on the processes and configuration shown above).

If the authenticity and identity check is successful, the SAM service deletes the Signer key pair referenced by the data it received through the request. More specifically, the SAM service requests the deletion of the Wrapped Key associated with the Signer key pair to be deleted from the KM. Otherwise, an error is returned.

Since Wrapped Keys do not contain keys in plaintext, they do not require any further destruction method.

Purpose	This operation is invoked by the requestor to delete a Signer key pair.		
Interface	SAM Peer Interface (see [IF_SPI])		
Method of use	The operation is called by client software according to the SAM peer interface specification (see [IF_SPI]).		
Inputs	**Parameter**	**Typ**	**Description**
	keyid	String	The ID of the key to be deleted

Output	Parameter	Typ	Description
		void	A result without error indicates a positive operation.

Error Messages	Typ	Reason
	ContainerParseError	If the input cannot be parsed into a message container
	MessageParseError	If the input data cannot be parsed into the specified message type
	UnknownMessageType	The contained message type was not known to the processor.
	UnknownCredentialType	The credential type included was not known to the processor.
	AuthenticationError	The credential could not be validated.
	UnknownUser	The credential could not be associated with a user.

	NotAuthorized	If the acting user is not authorized to perform the current operation If the acting user does not have access to the specified KeyID

	NotOperational	A component (i.e. database, HSM) is not available or not in an operative state.
	UnexpectedError	When an error condition occurs that was not anticipated
	NotFound	Wenn kein Schlüssel für die eingegebene keyid existiert

Security Audit Log Entry	In case of success An audit record is created that contains at least the following information ▪ the acting UserID ▪ the timestamp of the action ▪ the name of the action ▪ that the action was successful ▪ the UserID for which the key was removed ▪ the KeyID of the key pair that was deleted In case of an error The following errors issue regular log messages, but no audit log entries (since no acting user could be inferred): ▪ ContainerParseError ▪ UnknownCredentialType ▪ AuthenticationError ▪ UnknownUser Otherwise An audit record is created containing at least the following information ▪ the acting UserID ▪ the timestamp of the action ▪ the name of the action ▪ that the action failed ▪ a brief description of the cause of the failure

Table 21: Description of Delete_Signer_Key_Pair operation

7.1.6 SAM Peering Interface: Signing

The following Figure shows an overview of the process of creating a remote signature using the SAM.

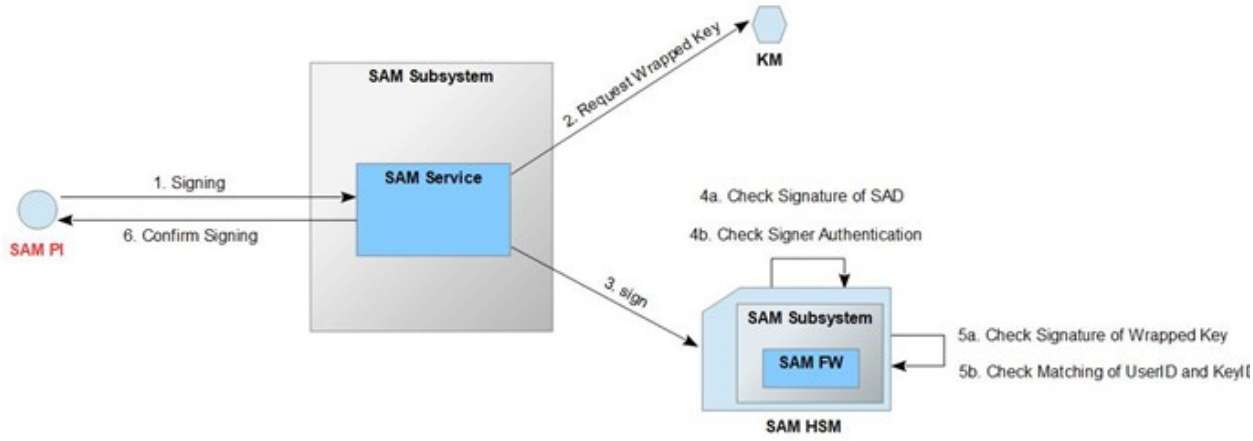


Figure 21: Overview of the Signing operation flow

A Signer uses the SAM peer interface to request the operation. Authentication and identification of the requesting Signer is required beforehand.

When the request is retrieved, the SAM service checks whether the requesting Signer is identified and authenticated.

If the check regarding identity and authenticity is successful, the SAM service requests the signing of the data to be signed, which is transferred with the received request. In detail, the SAM service requests

the wrapped key referenced in the SAD and

the signing of the data to be signed at the SAM HSM. The SAM firmware verifies that:

the authenticity of the Signer and the signature of the SAD are valid. the signature of the Wrapped Key is valid.

the user ID and the key ID of SAD of the Wrapped Key match.

If these steps are successful (validated), the data to be signed is signed at the SAM of the cryptographic module and returned to the SAM service. Otherwise, an "error" is returned.

The SAD basically has the following structure: {SignerAuthenticationData, KeyID, DTBS/R}. Reference] contains a more detailed ASN.1 definition of the structure.

Purpose	This operation is invoked by the Signer to create a signature using a qualified signature key.

Interface	SAM Peer Interface (see [IF_SPI])		
Method of use	The operation is called by a client software according to the specification of the SAM peer interface (see [IF_SPI]).		
Input	**Parameter**	**Type**	**Description**

	sad	Binary	The Signed ASN.1 structure contains the following components: ID token of the user KeyID of the key to be used for remote signing Hash value(s) to be signed Algorithm for signature generation
Output	**Parameter**	**Type**	**Description**
	sigbin	Binary	Signature binary returned by the cryptographic module.
Error messages	**Type**		**Reason**
	ContainerParseError		If the input cannot be parsed into a message container
	MessageParseError		If the input data cannot be parsed into the specified message type

UnknownMessageType	The contained message type was not known to the processor
UnknownCredentialType	The credential type included was not known to the processor
AuthenticationError	The credential could not be validated
UnknownUser	The credential could not be associated with a user
NotAuthorized	If the acting user is not authorized to perform the current operation If the acting user does not have access to the specified KeyID
NotOperational	A component (e.g. database, HSM) is not available or not in an operational state
UnexpectedError	When an error condition occurs that was not anticipated
NotFound	A key corresponding to the passed keyid does not exist.
InvalidAlgorithm	Specified signature algorithm does not match the key algorithm. Specified hash algorithm does not match digest size.
InvalidAuthentication	Identity token in SAD does not pass signature verification. UserID in the identity token does not match the UserID provided by the credential. Identification level is not sufficient (at least 'substantial').

	InvalidSignature	SAD signature cannot be verified with the public key embedded in the identity token.

Security Audit Log Entry	In case of success An audit record is created that contains at least the following information
	▪ the UserID of the acting user ▪ the timestamp of the action ▪ the name of the action ▪ that the action was successful ▪ the UserID of the user for whom the signature was created ▪ the KeyID of the signature key that was used to create the signature
	In case of an error
	The following errors emit regular log messages, but no audit log entries (since no acting user could be derived):
	▪ ContainerParseError ▪ UnknownCredentialType ▪ AuthenticationError ▪ UnknownUser
	Otherwise
	An audit record is created, containing at least the following information:

	▪ the UserID of the acting user ▪ the timestamp of the action ▪ the name of the action ▪ that the action failed ▪ a brief description of the cause of the failure

Table 22: Description of the Signing operation

7.2 SSA REST Interface

The REST interface of the SSA provides the following operations essential for server signatures:

Component	Operation	Description
-	(POST) /signing	Request to create a remote signature

Table 23: Operations of the SSA REST Interface

7.2.1 SSA REST Interface: Signing

Purpose	This operation is called to initiate the process of remote signing via the SSA.		
Interface	SSA REST Interface (siehe [IF_SSA])		
Method of use	The operation is called by a client software according to the specification as a POST method on the SSA REST interface (see [IF_SSA]).		
Input	Parameter	Type	Description
	sad	Binary	SAD used to create the remote signature

Output	Parameter	Type	Description
	signature(s)	String	Created signature(s), e.g.: { "123": "MIIHtQYJKoZIhvcNAQcC...dxteY7b0vcjCgk1aFnxd"}

Error messages	Type	Reason
	500	SsaException

Table 24: Operation /signing

7.3 SAK REST Interface

The REST interface of the SAK provides the following operations essential for server signatures:

Component	Operation	Description
Smartcard Operations Component	(POST) /smartcard/remote-signature/key	Request to create a key object
Digest Operations Component	(POST) /digest/create-signature (s)	Request creation of a signature via a local SAK /OS
	(GET) /digest/remote-signature/ {uuid}	Retrieval of SAD data for client side signature

	(POST) /digest/remote-signature/{uuid}	Adding the SAD signature in the remote signing process
	(POST) /digest/remote-signature /basic/sign	Request remote signing via a remote SAK
Trusted Checker Component	(POST) /trusted-checker/ check- document(s)	Verification of one or more documents before signature creation

Table 25: Operations of the SAK REST Interface

The operations listed are explained in more detail below with regard to their use, parameters and error cases.

7.3.1 SAK REST Interface: /smartcard/remote-signature/key

Purpose	This operation is called to obtain the creation of a key object for a remote signing process.		
Interface	SAK REST Interface (siehe [IF_SAK])		
Method of use	The operation is called by a client software according to the specification as a POST method on the SAK REST interface (see [IF_SAK]).		
Input	**Parameter**	**Type**	**Description**
	idToken	String	ID token of the requesting user
	keyId	String	KeyID of the signature key to be used for the remote signature process

	certificate	String	The certificate associated with the signature key
Output	Parameter	Type	Description
	keyObject	KeyObject	The created remote signature process relevant key object
Error messages	Type		Reason
	500		SecureFrameworkException

Table 26: Operation /smartcard/remote-signature/key

7.3.2 SAK REST Interface: /digest/create-signature(s)

Purpose	This operation is called to request the creation of a signature with a qualified signature key.		
Interface	SAK REST Interface (siehe [IF_SAK])		
Method of use	The operation is called by a client software according to the specification as a POST method on the SAK REST interface (see [IF_SAK]).		
Input	Parameter	Type	Description
	keyObject	keyObject	Signature key
	documents	Document[]	To Signer files
	proofReport	ProofReport	Check result of the files

	hashAlgorithm	HashAlgorithm	Hash algorithm to use
	signatureContainerFormat	String	Signature type
Output	Parameter	Type	Description
	signature(s)	String	Created signature(s), e.g.: { "123": "MIIHtQYJKoZIhvcNAQcC... dxteY7b0vcjCgk1aFnxd"}
Error messages	Type		Reason
	500		SecureFrameworkException

Table 27: Operation /digest/create-signature(s)

The 'Document' type has the following structure:

Component	Type	Description	JSON definition
id	String	Identifier (uuid) of the document to be checked	{"id":"[STRING]"}
fileName	String	Name of the document to be checked	{"fileName":"[STRING]" }
data	String	Data representation of the document to be checked	{"data":"[STRING]"}

Table 28: Datentyp 'Document'

The type 'ProofReport' has the following structure:

Component	Type	Description	JSON definition
checkedDocuments	checkDocument []	Checking information for a document to be checked	{"checkedDocuments":"[checkDocument[]]"}
creationTime	Number	Name of the document to be checked	{"creationtime":[NUMBER]}
signature	String	Data representation of the document to be checked	{"signature":"[STRING]"}

Table 29: Data type 'ProofReport'

The type 'HashAlgorithm' has the following structure:

Component	Type	Description	JSON definition
oid	String	OID of the hash algorithm to be used	{"oid":"[STRING]"}
name	String	Name of the hash algorithm to be used	{"name":"[STRING]"}

Table 30: Datentyp 'HashAlgorithm'

7.3.3 SAK REST Interface: /digest/remote-signature/{uuid}

Purpose	This operation is called to initiate the retrieval of SAD-TBS data for client-side signature.
Interface	SAK REST Interface (siehe [IF_SAK])

Method of use	The operation is called by a client software according to the specification as a GET method on the SAK REST interface (see [IF_SAK]).		
Input	Parameter	Type	Description
	uuid	String	The process ID from the key object used in the process
Output	Parameter	Type	Description
	sad-data	String	The SAD data to be signed
Error messages	Type		Reason
	500		SecureFrameworkException

Table 31: Operation /digest/remote-signature/{uuid}

7.3.4 SAK REST Interface: /digest/remote-signature/{uuid} 2

Purpose	This operation is called to add a created SAD signature to the associated remote signature process and thus trigger it.		
Interface	SAK REST Interface (siehe [IF_SAK])		
Method of use	The operation is called by a client software according to the specification as a POST method on the SAK REST interface (see [IF_SAK]).		
Input	Parameter	Type	Description

	uuid	String	The process ID from the key object used in the process
Output	Parameter	Type	Description
		204	A result without errors indicates a positive operation (SAD signature was added to the remote signature)
Error messages	Type		Reason
	500		SecureFrameworkException

Table 32: Operation /digest/remote-signature/{uuid}

7.3.5 SAK REST Interface: /digest/remote-signature/basic/sign

Purpose	This operation is called to initiate the process of remote signing via the SAK.		
Interface	SAK REST Interface (siehe [IF_SAK])		
Method of use	The operation is called by a client software according to the specification as a POST method on the SAK REST interface (see [IF_SAK]).		
Input	Parameter	Type	Description
	sad	Binary	SAD used to create the remote signature
Output	Parameter	Type	Description

	signature(s)	String	Created signature(s), e.g.: { "123": "MIIHtQYJKoZIhvcNAQcC...dxteY7b0vcjCgk1aFnxd"}
Error messages	Type		Reason
	500		SecureFrameworkException

Table 33: Operation /digest/remote-signature/basic/sign

7.3.6 SAK REST Interface: /trusted-checker/check-document(s)

Purpose	This operation is called to prevalidate one or more documents before the actual signature creation.		
Interface	SAK REST Interface (siehe [IF_SAK])		
Method of use	The operation is called by a client software according to the specification as POST method on the SAK REST interface (see [IF_SAK]).		
Input	Parameter	Type	Description
	document	Document	Data type containing information about the document to be signed (see below).
Output	Parameter	Type	Description
	proofReport	ProofReport	Data type containing the generated proof report (see below).

Error messages	Type	Reason
	500	SecureFrameworkException

Table 34: Operation /trusted-checker/check-document(s)

The type 'Document' has the following structure:

Component	Type	Description	JSON definition
id	String	Identifier (uuid) of the document to be checked	{"id":"[STRING]"}
fileName	String	Name of the document to be checked	{"fileName":"[STRING]"}
data	String	Data representation of the document to be checked	{"data":"[STRING]"}

Table 35: Data type 'Document'

The 'ProofReport' type has the following structure:

Component	Type	Description	JSON definition
checkedDocuments	checkDocument []	Checking information for a document to be checked	{"checkedDocuments":"[checkDocument[]]"}
creationTime	Number	Name of the document to be checked	{"creationtime":[NUMBER]}
signature	String	Data representation of the document to be checked	{"signature":"[STRING]"}

Table 36: Data type 'ProofReport'

The type 'checkDocument' has the following structure:

Component	Type	Description	JSON definition
documentId	String	Identifier (uuid) of the document to be checked	{"documentId":"[STRING]"}
errorMessage	String	Possible error messages regarding the performed test	{"errorMessage":[STRING]}
hashValue	String	Hash value of the checked document	{"hashValue":"[STRING]"}
mimeType	String	mimeType of the checked document	{"mimeType":"[STRING]"}
proofResult	Boolean	Result of the proof report	{"proofResult":[BOOLEAN]}

Table 37: Data type 'checkDocument'

7.4 Key Manager REST Interface

The Key Manager REST interface provides the following operations essential for server signatures:

Component	Operation	Description
KeyListResource	(GET) /private-keys	Query active private remote signature keys
ObjectListResource	(GET) /objects	Querying management objects (e.g. Signer)

-	(POST) /objects	Registration of a new managed object

Table 38: Operations of the Key Manager REST Interface

7.4.1 Key Manager REST Interface: /private-keys

The operations listed are explained in more detail below with regard to their use, parameters and error cases.

Purpose	This operation is called to query the objects of the PRIVATE_KEY type with the ACTIVE status present in the Key Manager.		
Interface	Key Manager REST Interface (siehe [IF_KM])		
Method of use	The operation is called by a client software according to the specification as a GET method on the Key Manager REST interface (see [IF_KM]).		
Input	**Parameter**	**Type**	**Description**
	authorization	String	ID token in the format: "Bearer: MII..."
	return	Object	Specifies the data fields to be returned with respect to found objects. If this parameter is omitted, all data fields will be returned
	id	String	Object identifier as query parameter for filtering the result

	state	String	Object status as query parameter to filter the result Possible values: PRE_ACTIVE, ACTIVE, DEACTIVATED, COMPROMISED, DESTROYED, DESTROYED_COMPROMISED
	objecttype	String	Objekttyp als Query-Parameter zur Filterung des Ergebnisses. Possible values: CERTIFICATE, CERTIFICATE_REQUEST, OPAQUE_OBJECT, PGP_KEY, PRIVATE_KEY, PUBLIC_KEY, SECRET_DATA, SPLIT_KEY, SYMMETRIC_KEY, X_USER, X_USER_GROUP
	attribute-params	Object	Attributes as query parameters to filter the result Usage: $AttributeName;$QueryExpression=$FilterValue [; $QueryExpression=$FilterValue] Example: name;contains=service&serialnumber; startsWith=1234&fresh=false
	objecttype	String	Type of the found object
	state	String	Status of the found object
	attributes	Attributes	Attributes of the found object
	dateattributes	DateAttributes	Validity period of the found object

Error messages	Type	Reason
	400	Invalid request {"type": "string", "title": "string", "detail": "string", "instance": "string", "status": "string", "additionalData": {}}
	401	Unauthorized access to the resource {"type": "string", "title": "string", "detail": "string", "instance": "string", "status": "string", "additionalData": {}}
	403	Access to the resource forbidden {"type": "string", "title": "string", "detail": "string", "instance": "string", "status": "string", "additionalData": {}}
	500	Internal server error {"type": "string", "title": "string", "detail": "string", "instance": "string", "status": "string", "additionalData": {}}

Table 39: Operation /private-keys

The 'Attributes' type has the following structure:

Component	Type	Description	JSON definition
comment	String	Comment	{"comment":"[STRING]"}
name	String	Attribute name	{"name":"[STRING]"}

Table 40: Data type 'Attributes'

The 'DateAttributes' type has the following structure:

Component	Type	Description	JSON definition
activation	String	Date of activation	{"activation":"[STRING]"}
not-after	String	Date of revocation	{"not-after":"[STRING]"}

Table 41: Data type 'DateAttributes'

7.4.2 Key Manager REST Interface: (GET) /objects

Purpose	This operation is called to query for managed objects at the Key Manager.		
Interface	Key Manager REST Interface (siehe [IF_KM])		
Method of use	The operation is called by a client software according to the specification as a GET method on the Key Manager REST interface (see [IF_KM]).		
	authorization	String	ID token in the format: "Bearer: MII..."
	return	Object	Specify fields to be returned for managed objects that are found. If this parameter is omitted, all fields of the managed objects will be returned. Usage: return=id&return=value&return=attributes&return=activation
	id	String	Object identifier for the management object to be filtered by

	state	String	Status of the managed objects to be filtered by Possible values: PRE_ACTIVE, ACTIVE, DEACTIVATED, COMPROMISED, DESTROYED, DESTROYED_COMPROMISED
	object type	String	Object type of the management object to be filtered by Possible values: CERTIFICATE, CERTIFICATE_REQUEST, OPAQUE_OBJECT, PGP_KEY, PRIVATE_KEY, PUBLIC_KEY, SECRET_DATA, SPLIT_KEY, SYMMETRIC_KEY, X_USER, X_USER_GROUP
	attribute-params	Attributes	Attributes as query parameters for filtering Usage: $AttributeName;$QueryExpression=$filtervalue
Output	Type		Description
	200		Successfull response {"id":"string","value":"string","objecttype":"string","state":"string","attributes":{"comment":"string","name":[string]},"dateattributes {"activation":"string","not-after":"string"}}
Error messages	Type		Reason
	400		Invalid request {"type": "string", "title": "string", "detail": "string", "instance": "string", "status": "string", "additionalData": {}}

	401	Unauthorized access to the resource {"type": "string", "title": "string", "detail": "string", "instance": "string", "status": "string", "additionalData": {}}
	403	Access to the resource forbidden {"type": "string", "title": "string", "detail": "string", "instance": "string", "status": "string", "additionalData": {}}
	500	Internal server error {"type": "string", "title": "string", "detail": "string", "instance": "string", "status": "string", "additionalData": {}}

Table 42: Operation (GET) /objects

7.4.3 Key Manager REST Interface: (POST) /objects

Purpose	This operation is called to register a new managed object (certificate) at the Key Manager.
Interface	Key Manager REST Interface (siehe [IF_KM])

Method of use	The operation is called by a client software according to the specification as a POST method on the Key Manager REST interface (see [IF_KM]).		
Input	Parameter	Type	Description
	authorization	String	ID token in the format: "Bearer: MII..."

	id	String	Object identifier of the management object to register
	value	String	Value of the management object to register
	object type	String	Object type of the management object to be registered Possible values: CERTIFICATE, CERTIFICATE_REQUEST, OPAQUE_OBJECT, PGP_KEY, PRIVATE_KEY, PUBLIC_KEY, SECRET_DATA, SPLIT_KEY, SYMMETRIC_KEY, X_USER, X_USER_GROUP
	attributes	Attributes	Attributes of the management object to be registered
	links	Link []	Link between managed objects (refers to the linked object ID, a link type describes the object relationship)
	contentHint	String	Note on the content of the object
Output	**Type**		**Description**
		200	Successful response {"id": "string", "value": "string", "objecttype": "string", "state": "string", "attributes": {"comment": "string", "name": ["string"]}, "dateattributes": { "activation": "srtring", "not-after": "string"}}
Error messages	**Type**		**Reason**

	400	Invalid request {"type": "string", "title": "string", "detail": "string", "instance": "string", "status": "string", "additionalData": {}}
	401	Unauthorized access to the resource {"type": "string", "title": "string", "detail": "string", "instance": "string", "status": "string", "additionalData": {}}
	403	Access to the resource forbidden {"type": "string", "title": "string", "detail": "string", "instance": "string", "status": "string", "additionalData": {}}
	500	Internal server error {"type": "string", "title": "string", "detail": "string", "instance": "string", "status": "string", "additionalData": {}}

Table 43: Operation (POST) /objects

The 'Attributes' type has the following structure:

Component	Type	Description	JSON definition
comment	String	Comment	{"comment":"[STRING]"}
name	String[]	Attribute name	{"name":"[STRING]"}
activation	String	Date of activation	{"activation":"[STRING]"}
not-after	String	Date of revocation	{"not-after":"[STRING]"}

Table 44: Data type 'Attributes'

The 'Link' type has the following structure:

Component	Type	Description	JSON definition
id	String	Comment	{"id":"[STRING]"}
type	String	Attribute name	{"type":"[STRING]"}

Table 45: Data type 'Link'

7.5 IdP REST Interface

The Identity Provider (IdP) REST interface provides the following operations essential for server signatures:

Component	Operation	Description
-	(POST) /id-token/sad	Embedding a public key as extension into the ID token

Table 46: Operations of the IdP REST Interface

The operations listed are explained in more detail below with regard to their use, parameters and error cases.

7.5.1 IdP REST Interface: /id-token/sad

Purpose	This operation is called to embed the public key passed with the request as an extension in the Id token.
Interface	IdP REST Interface (siehe [IF_IDP])
Method of use	The operation is called by a client software according to the specification as POST method on the IdP REST interface (see[IF_IDP]).

Input	Parameter	Type	Description
	idToken	String	The ID token to be extended by the public part of the SAD Signing key.
	sadPublicKey	String	The public key by which the ID token is to be extended.
Output	Parameter	Type	Description
	idTokenMod	String	The ID token extended by the public part of the SAD Signing key.
Error messages	Type		Reason
	400		Invalid request ("invalid request data")
	404		Unauthorized access to the resource ("authenticated session not available")
	500		Internal server error ("unexpected internal error")

Table 47: Operation /id-token/sad

7.6 Interface Specifications: Remote Signature API

The Remote Signature API provides the following operations essential for server signatures:

Operation	Description

login	Login to initalize the process requesting the generation of a remote signature gaining access to the operations to do so.
sign	Request to generate remote signature.

Table 48: Operations of the Remote Signature API

The Remote Signature API is available in two variants: JavaScript based and Java based.

The operations listed above are explained in more detail below with regard to their use, parameters and error cases.

7.6.1 Remote Signature API: login

Purpose	This operation is called to start the process of creating a remote signature, more precisely, to log in with an ID token to gain access to the operations necessary to do so.		
Interface	Remote Signature API (see [IF_RSAPI_JS] and/or [IF_RSAPI_JV])		
Method of use	The operation is called by a client software according to the specifion on the Remote Signature API (see [IF_RSAPI_JS] and/or [IF_RSAPI_JV]).		
Input	**Parameter**	**Type**	**Description**
	idToken	Binary String	The ID token to be extended by the public part of the SAD Signing key.
Output	**Parameter**	**Type**	**Description**
	RemoteSigner	Object	Remote signature creation API providing the sign operation.

Error messages	Type	Reason
	-	RemoteSignatureException

Table 49: Operation login

7.6.2 Remote Signature API: sign

Purpose	This operation is called to request the generation of a remote signature.		
Interface	Remote Signature API (see [IF_RSAPI_JS] and/or [IF_RSAPI_JV])		
Method of use	The operation is called by a client software according to the specifion on the Remote Signature API (see [IF_RSAPI_JS] and/or [IF_RSAPI_JV]).		
Input	Parameter	Type	Description
	hashAlgorithm	HashAlgorithm String	The hash algorithm to be used within the remote signatures process (e.g.: "sha256", "sha384", "sha512").
	keyId	String	KeyID of the signature key to be used for the remote signature process.
	hashValues	Binary String[]	The hash values of the data to be signed in the remote signature process.
Output	Parameter	Type	Description

	signature(s)	String	Created signature(s), e.g.: { "123": "MIIHtQYJKoZIhvcNAQcC...dxteY7b0vcjCgk1aFnxd"}
Error messages	Type		Reason
	-		RemoteSignatureException

Table 50: Operation login

7.7 SAM Management Command Line Interface

The Administrators are allowed to execute the following management operations on the SAM subsystem per manageSAM.sh or manageFW.sh script:

Operation	Command
Start	manageSAM [-v] start <instanceId>
Stop	manageSAM [-v] stop <instanceId>
Check Code Integrity	manageSAM [-v] integrity code <instanceId>
Check Data Integrity	manageSAM [-v] integrity data
Initialize SAM Firmware	manageFW [-v] init LogonPass=<user>,<credential> <MBK slot id>
Set IdP Public Keys	manageFW [-v] setidpkeys LogonPass<user>,<credential> <certfile>

Table 51: SAM Maintenance Operations

The following figure shows an overview of the management operations carried out by the module SAM Management Module.

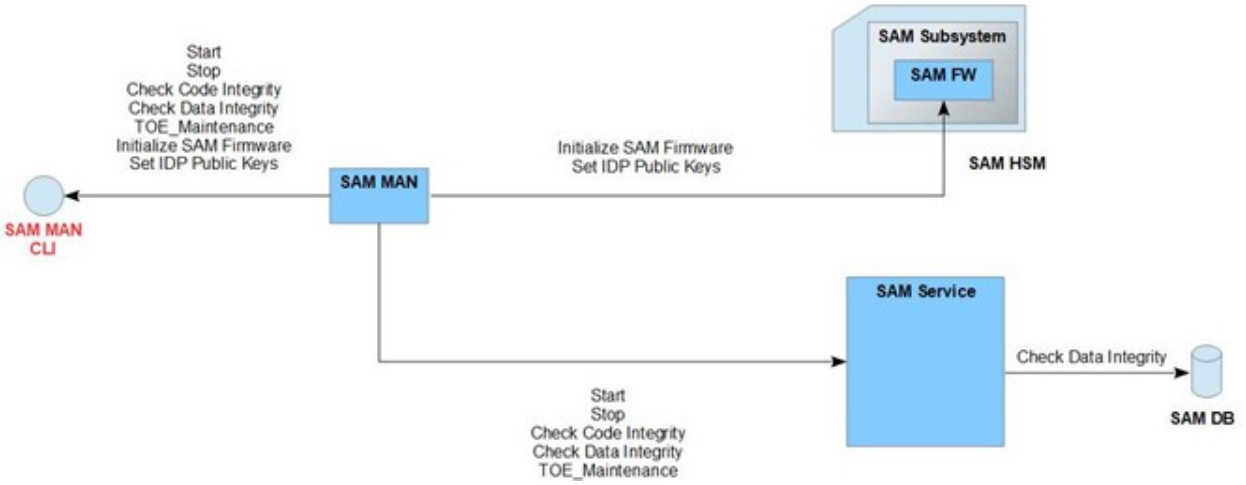


Figure 22: Overview of SAM Maintenance operations

The most common success and error scenarios for calling the management operations per manageSAM.sh and manageFW.sh script are listed below:

Operation	Console Output (Exit Code)	Action
Start	XYZ code integrity check … ok The subsystem “sam<instanceId>” has been started … ok (0)	-
Start	The subsystem “sam<instanceId>” is already running … warning (0)	-
Start	XYZ code integrity check … failed Starting the subsystem “sam<instanceId> has been abandoned ... failed (1)	Check for modifications.
Stop	The subsystem “sam<instanceId>” has been stopped … ok (0)	-

Stop	The subsystem “sam<instanceId>” is not running … warning (0)	-
Check Code Integrity	XYZ code integrity check … ok (0)	-
Check Code Integrity	XYZ code integrity check … failed. (1)	Check deployment for modifications.
Check Data Integrity	Data integrity check … ok (0)	-
Check Data Integrity	Data integrity check … failed (1)	Investigate cause. Check database for modifications.
Initialize SAM Firmware	Initialization of SAM Firmware … ok (0)	-
Initialize SAM Firmware	Initialization of SAM Firmware … failed (1)	Investigate cause.
Set IdP Public Keys	Import of IdP Public Keys … ok (0)	-
Set IdP Public Keys	Import of Public Keys … failed (1)	Investigate cause.

Table 52: Calling of management operations per manageSAM.sh and manageFW.sh

8 Format Specifications

8.1 SAD Structure

The Signature Activation Data (SAD) is a data structure that contains the following components:

- the Identity Token of the user
- the KeyID of the key to be used for the server signature
- the hash value to be used for signatures
- the hash algorithm used for signature generation

The SAD have the following simplified structure: {SignerAuthenticationData, KeyID, DTBS/R, Algorithm}. Below is the more detailed ASN.1 definition of the SAD data structure used.

```
-- SAD

SADTBS ::= SEQUENCE {

identityToken   SignedIdentityToken,

keyid   Directory String,

hashAlgorithm   AlgorithmIdentifier,

tbsHashValues   SEQUENCE OF OCTET STRING

}

SAD ::= SEQUENCE {

tbs SADTBS,

signature   OCTET STRING

}
```

8.2 ID Token Structure

The identity token (ID token) is the authorization type used to authenticate users of the server signing.

The ID token contains the user ID, the role of the user is determined from the information in the SAM database. In case of a Privileged User Technical, his role can also be determined based on a signature for which an X.509 certificate is stored in the SAM. The role is checked in the respective operation. An identity token of an unregistered user or the signature of an unregistered X.509 certificate is not accepted.

Below is the more detailed ASN.1 definition of the ID token data structure used.

```
-- Identity Token

SignedIdentityToken ::= SEQUENCE {

data    CompactIdentityToken,

Signer  IssuerSerial, signatureAlgorithm    AlgorithmIdentifier, signature  OCTET
STRING,

type    OBJECT IDENTIFIER   OPTIONAL

}

CompactIdentityToken ::= SEQUENCE {

userId  Directory String,

notBefore   Generalized Time,

notAfter    Generalized Time, groupAssociations SET OF CompactGroupAssiocation,
extensions [0] IMPLICIT CompactExtensions

}

CompactExtensions ::= SET OF CompactExtension CompactExtension ::= SEQUENCE {

oid OBJECT IDENTIFIER,

data    ANY DEFINED BY oid

}

CompactRoleAssociation ::= CHOICE {

shortRoleAssocaition    CompactShortRoleAssociation, longRoleAssocation
CompactLongRoleAssocation

}
```

```
CompactShortRoleAssociation ::= SEQUENCE { shortId  OCTET STRING, extensions [2]
IMPLICIT CompactExtensions

}

CompactLongRoleAssocation ::= SEQUENCE {

app [0] Directory String,

role    [1] Directory String, extensions [2] IMPLICIT CompactExtensions

}

CompactGroupAssociation ::= SEQUENCE {

id  OCTET STRING,

roleAssociations    SET OF CompactRoleAssiocation, extensions [0] IMPLICIT
CompactExtensions

}
```

9 References

AGD_OPE	proNEXT SignatureActivationModule – Operational User Guide, Version 1.3, October 2020
AGD_PRE	proNEXT SignatureActivationModule – Installation Guide, Version 1.3, October 2020
CCert CP5	TÜV Rheinland Nederland B.V.Headoffice: Certification Report CryptoServer CP5 Se12 5.1.0.0, CryptoServer CP5 Se52 5.1.0.0, CryptoServer CP5 Se500 5.1.0.0, CryptoServer CP5 Se1500 5.1.0.0, May 2020
CCert SAM	TÜV Informationstechnik GmbH: QSCD-Certificate proNEXT SignatureActivationModule 1.0.0, December 2020
CCert SAK	TÜV Informationstechnik GmbH: Signaturanwendungskomponente proNEXT Secure Framework 2.0, July 2016
EN419401	Electronic Signatures and Infrastructures (ESI); General Policy Requirements for Trust Service Providers. V2.2.1, April 2018
EN319411-1	Electronic Signatures and Infrastructures (ESI); Policy and security requirements for Trust Service Providers issuing certificates; Part 1: General requirements. V1.2.2, April 2018
EN319411-2	Electronic Signatures and Infrastructures (ESI); Policy and security requirements for Trust Service Providers issuing certificates; Part 2: Requirements for trust service providers issuing EU qualified certificates. V2.2.2, February 2016
EN419221-5	Protection Profiles for TSP cryptographic modules – Part 5: cryptographic module for Trust Services. English version EN 419221-5:2018

EN419241-1	Vertrauenswürdige Systeme, die Serversignaturen unterstützen - Teil 1: Allgemeine Systemsicherheitsanforderungen; Deutsche Fassung EN 419241-1:2018
EN419241-2	Vertrauenswürdige Systeme, die Serversignaturen unterstützen – Teil 2: Schutzprofil für qualifizierte Signaturerstellungseinheiten zur Serversignierung; Deutsche Fassung EN 419241-2:2019
FSP_TDS	proNEXT SignatureActivationModule – TOE Specification. Version 1.2, October 2020
IF_IDP	Interface description of the proNEXT IdP. Version 3.2.0, February 2021
IF_KM	Interface description of the proNEXT Key Manager. Version 2.0, February 2021
IF_RSAPI_JS	Interface description of the JavaScript based Remote Signature API, October 2021
IF_RSAPI_JV	Interface description of the Java based Remote Signature API, October 2021
IF_SAK	Interface description of the proNEXT Secure Framework. Version 1.11.0, February 2021
IF_SPI	Interface description of the proNEXT SAM Peering Interface. February 2021
IF_SSA	Interface description of the proNEXT Server Signing Application. February 2021

ISO32000-1	Document management - Portable document format - Part 1: PDF 1.7. Version ISO 32000-1: 2008, July 2008
InstCP5	Installation Guide - CryptoServer Se-Series Gen2 CP5, Version 1.0, May 2020
KMIPv20	Key Management Interoperability Protocol Specification. Version 2.0, October 2019
RFC2315	PKCS #7: Cryptographic Message Syntax. Version 1.5, March 1998
ST_SAM	proNEXT SignatureActivationModule – Security Target, Version 1.4, October 2020
ST_SF	proNEXT Secure Framework – Security Target, Version 1.14, September 2016

10 Abbreviations

Abbreviation	Term
AM	Audit Manager
API	Application Program Interface
ASN	Abstract Syntax Notation
CA	Certification Authority
CC	Common Criteria
CP5	Utimaco CP5 HSM
DIN	Deutsches Intistut für Normung
DTBS	Data to be signed
DTBS/R	Data to be signed Representation
eID	Electronic Identification
eIDAS	Electronic Identification, Authentication and Signature
EN	European Norm
ETSI	European Telecommunications Standards Institute
HSM	Hardware Security Module

ID	Identifier
IdP	Identity Provider
IF	Interface
JSON	Java Script Object Notation
KLMS	Key Lifecycle Management System
KM	Key Manager
KMIP	Key Management Interoperability Protocol
LSP	Local Service Provider
NTP	Network Time Protocol
OCSP	Online Certificate Status Protocol
OID	Object Identifier
OS	Operating System
PDF	Portable Document Format
PID	Person Identification Data
PIN	Personal Identification Number

PP	Protection Profile

QSCD	Qualified electronic Signature/Seal Creation Device
RA	Registration Authority
REST	Representational State Transfer
RM	Registration Manager
SAD	Signature Activation Data
SAK	Signaturanwendungskomponente
SAK/OS	Signaturanwendungskomponente / Operating System
SAM	Signature Activation Module
SAP	Signature Activation Protocol
SCA	Signature Creation Application
SCAL	Sole Control Assurance Level
SCDev	Signature Creation Device
SIC	Signature Interaction Component
SSA	Server Signing Application

SSL	Secure Socket Layer
SSSrv	Server Signing Service
TBS	To Be Signed
TSP	Trusted Service Provider
TW4S	Trusted system supporting server signatures
UC	Use Case
UI	User Interface
UM	User Manager
US	Usage Scenario
UUID	Universally Unique Identifier
VA	Validation Authority
X.509	Standard defining the format of public key certificates