securosys

SWISS SECURITY TECHNOLOGIES
FOR COMMUNICATIONS SYSTEMS

Primus HSM Audit & Attestation

Usage with Qualified Trust Service Providers

Application Note

Edition 1 (v1.0.2)

Published: 2023-5

Securosys SA, Max-Högger-Strasse 2,
8048 Zürich , Switzerland
Tel. +41 44 552 31 00 • www.securosys.com
info@securosys.com

Document Information and Revision Control

Version	Date	Author	Description, Changes
1	31.10.2022	PM	Initial document
1.0.1	23.11.2022	PM	Added CLI samples using JCE
1.0.2	29.05.2023	PM/MS	Corrected Proxy connection parameters for PrimusTool and KeytoolX

File: PrimusHSM_AuditAndAttestation_AN-E01.docx

Table of Contents

1 Introduction

Compliance and security policies for e.g., digital identity applications require the certificate keys to be issued in a trusted way. This procedure is called a key ceremony. It is standard for any certificate authorities or qualified trust service providers issuing certificates for e.g., qualified seal or signature creation. Traditionally the key ceremony is witnessed in person and the key creation environment, e.g., HSM, and the key creation process is audited. All of this is costly and unscalable.

The Primus HSM offers with the attestation feature a way to provide cryptographic evidence of all relevant attributes and allows an auditor to verify it at any time and without physical presence. The attestation is delivered by a signed export file of the relevant data, key attributes, HSM configuration, diagnostics, and logs. The signed attestation files can be verified by the underlaying trust chain which goes back to the Securosys root certificate which is only inherent in any Primus HSM.

1.1 Target Audience

This document describes the work procedure to conduct a key ceremony based on the attestation feature of Primus HSM and is intended for qualified trust service providers, auditors and organizations requesting a certificate for e.g., qualified signature or seal creation.

1.2 Glossary

Acronym	Definition
AN	Securosys Application Note
CA	Certification Authority
CC	Common Criteria for IT Security Evaluation
CloudsHSM	HSM as a service, operated by Securosys
C(E)RT	Certificate
CSR	Certificate Signing Request
Decanus	Decanus Terminal for remote administration of Primus HSM devices or partitions
FIPS	Federal Information Processing Standard
HSM	Hardware Security Module (physical or as a service)
PKI	Public Key Infrastructure
QTSP	Qualified Trust Service Provider
RSA	Rivest, Shamir, & Adleman (public key encryption technology)
SO - Security Officer	Management role which is split to multiple operators. To activate 2 must be authenticated.
UG	Securosys User Guide
XML	Extensible Markup Language, defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. Format used for HSM configuration, user creation and attestation files.

Table 1 Glossary

1.3 References and More Information

[1] Primus HSM User Guide for v2.10/v2.8 Edition 11 or later

[2] https://www.openssl.org/docs/

All Securosys documentation is downloadable from the Securosys Support Portal.

1.4 Support Contact

If you encounter a problem while using the key attestation feature, make sure that you have read the referenced documentation. If you cannot resolve the issue, contact your supplier or Securosys Customer Support. The Securosys Support Portal for registered users is reachable under https://support.securosys.com.

2 Audit and Attestation Concept

The Primus HSM attestation feature is based on a HSM internal certificate chain. This trust chain is going back to an HSM inherent root certificate issued by Securosys. The Securosys root certificate is common in all Securosys Primus HSMs. The intermediate certificate is identifying the device; a specific audit key is used for any device attributes and the attestation key is per partition. The Securosys HSM attestation root key certificate is published on the Securosys support portal.

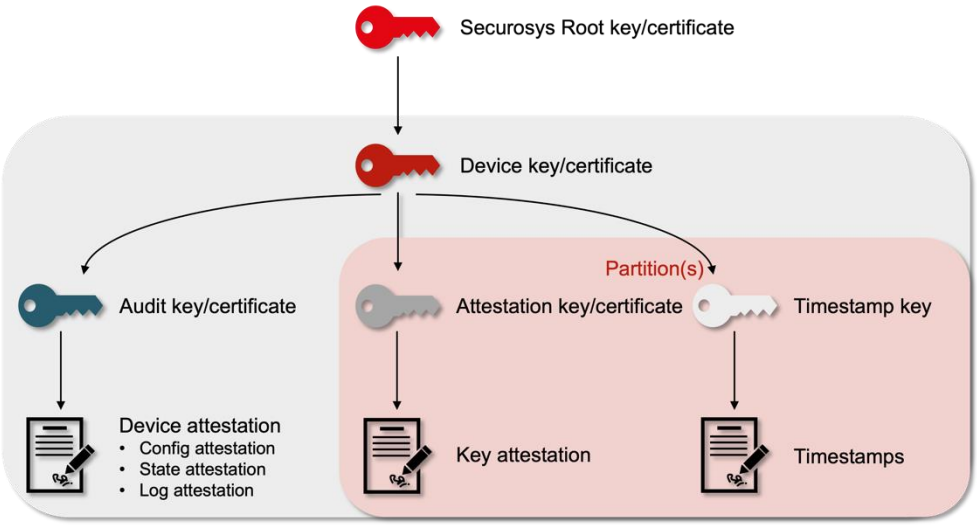

Figure 1 Certificate Chain Details of Primus HSM Attestation

Using the above keys 4 different signed XML attestation files can be exported, including the full chain of trust.

Attestation File	Level	Evidence	User / Interface
Key attestation	Partition	Key in HSM Key attributes, flags, parameters	API / Data Interface
Config attestation	Device	Syslog configuration Network configuration Device & partition configuration Management setup Roles / Users	SO / Management (CLI/Decanus)
State attestation	Device	Hardware model Software version Operation mode (FIPS/CC) Cluster size	SO / Management (CLI/Decanus)
Log attestation	Device	System status Key creation Key usage Configuration changes Logins	SO / Management (CLI/Decanus)

Table 2 Certificate Chain Details of Primus HSM Attestation

3 Audit and Attestation Procedure

The audit and attestation procedure are described using the example of the issuance of a certificate for a signing key by a Qualified Trust Service Provider to an organization, for the purpose of e.g., code signing, or qualified signing or sealing of documents conforming to the eIDAS regulation.

For convenience and the ease of understanding, we will use `keytoolX` – Primus JCE Provider adapter to Java's keytool – in this application note to generate a generic signing key. One can however also use their own application of choice here.

To guarantee independence of Securosys, the verification procedure is based on the OpenSSL software library, which is available as opensource software.

Note that although the audit and attestation procedure described in this application note are based on the issuance of a certificate for a signing key, the procedure itself is by nowhere limited to this and can equally be applied to other use cases like the creation of a key for a Root or Issuing CA.

The audit and attestation procedure involves the following steps, that are described in detail in the following chapters:

1 The creation of the Key Attestation File (chapter 4)
2 Validation of the Key Attestation File (chapter 5)
3 Device Attestation (Audit) (chapter 6)
4 Verification of the Signed Audit Information (chapter 7)

The following parties are involved in the audit and attestation procedure:

- HSM operation (SO role)
- Organization (HSM user, signing service operator)
- Qualified Trust Service Provider (Issuer of certificate for code signing or qualified signing / sealing of documents)
- Auditor

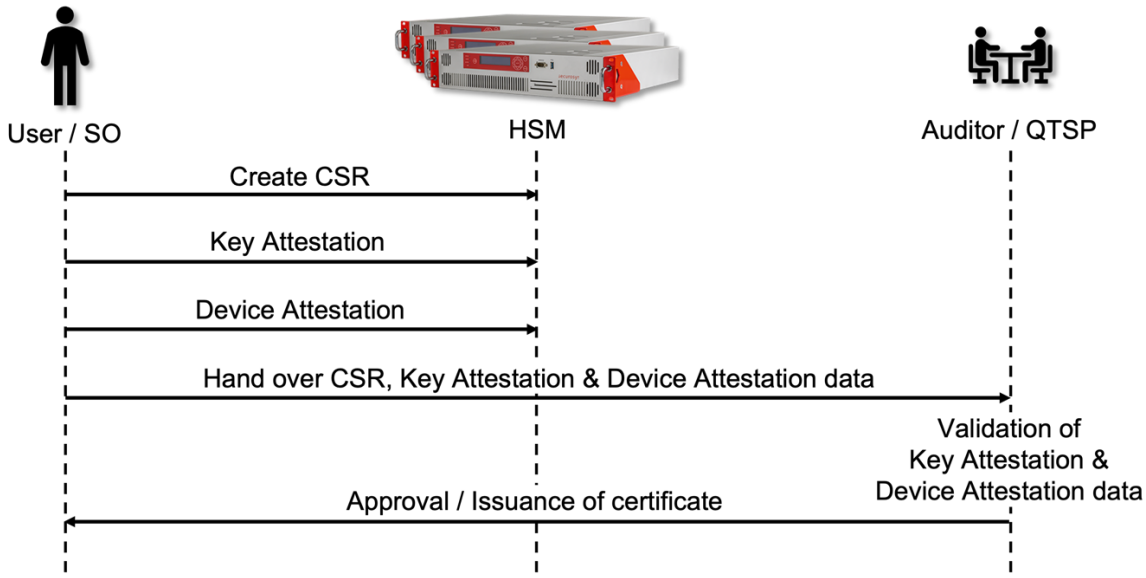

Figure 2 Audit & Attestation Procedure

3.1 Preparation of Primus HSM Attestation Feature

The attestation feature must be enabled by the SO role before it can be used by any user. The activation of the attestation feature involves the following steps:

1 Obtain the license to use the Attestation feature from Securosys and install it

2 Install RootKey and RootKey Store

Caution: this step erases any previously installed Intermediate Key Store!

SYSTEM -> ROOT KEY ELEMENT -> INSTALL RKS/RNG  

`hsm_sec_install_rke`

- Insert the USB stick with the license file and select the appropriate license
- Enter the Genesis PIN
- The procedure should complete with the message “Root key store setup successful!”

3 Create device attestation and audit key as SO via management
Caution: this erases any previously installed Intermediate Key!

SYSTEM -> ROOT KEY ELEMENT -> SETUP RKS 

`hsm_sec_setup_rks`

- The procedure should complete with the message “Root key store setup successful!”

All CloudsHSM devices are already enabled for attestation usage and hence the afore mentioned steps can be ignored for CloudsHSM users.

Please note, that the root key store must be setup first, prior any attestation functionality like key-, device-, partition-, and log-attestation (e.g., to export signed security logs) can be used.

More details on the HSM configuration are provided in the Primus HSM User Guide [1].

3.2 Prerequisite

Before you start, download the following Securosys tools and certificates from the Securosys Support Portal and copy them to your working directory for the further procedures.[1]

- primus-tools.jar (v2.1.4 or later)
- keytoolX.jar (2019-10-04 or later)
- primusX.jar (v1.8.6 or later)
- Securosys Primus Root Certificate, available in the Knowledge Base under “01. General Information” → “2 General Product Information”
  - Developer Program: `primusedev_rsa_root_dev.crt`
  - Primus HSM / CloudsHSM: `hsm_rsa_root.crt`

The verification procedure is based on opensource software library OpenSSL to guarantee independence of Securosys.

- E.g., LibreSSL (v2.8.3 or later), OpenSSL (v3.0 or later) or equivalent

[1] These tools are not required incase the REST API is used for key creation / key attestation. See also chapter 4 and the examples in appendix 0.

4 Key Attestation Procedure

Provided the HSM is enabled for attestation, the organization (HSM user) creates a signing key, prepares a CSR and the corresponding Key Attestation File in the following sequence.

1. Create Key Attestation key (chapter 4.1)
2. Create Signing key (chapter 4.2)
3. Generate CSR (chapter 4.3)
4. Export Key Attestation (chapter 4.4)
5. Submit CSR and Key Attestation File to the QTSP (chapter 4.5)

For ease of understanding, the command samples are shown for the `primus-tools` and `keytoolX`. A CLI sample using JCE is available in appendix Key Attestation JCE CLI Sample. A java-code sample for performing Key Attestation via the JCE API is provided in appendix Key Attestation JCE API Code Sample and examples for using the REST API are provided in the appendix Key Attestation REST API Sample.

The following command samples are provided for Developer Program or on-premises HSM setups.

For CloudsHSM, the proxy user and password parameters (`keytoolX: -proxyuser / -proxypassword; primus-tools: -primusproxyuser / -primusproxypassword`) must be added in the command, e.g.

```
java -jar keytoolX.jar -host <HsmUrlorIpAddr> -port <port> -user
<PartitionName> -password <HsmUserPassword> -proxyuser <ServiceUserName>
-proxypassword <ServiceUserPassword>
```

4.1 Create Key Attestation Key

Create a Key Attestation key on the user partition by running the following command:
Help for parameter type & size is in the Appendix, see Key Attestation JCE CLI Sample

```
java -jar primus-tools.jar CreateAttestationKey -host <HsmUrlorIpAddr> -
port <port> -user <PartitionName> -password <HsmUserPassword> -keyname
<KeyAttestationKey> -type <type> -size <size>
```

Verify the correct creation of the Key Attestation key by either checking the log files on the HSM or running the following command that lists all key store objects:

```
java -jar primus-tools.jar ListKeyStoreObjects -host <HsmUrlorIpAddr>
-port <port> -user <PartitionName> -password <HsmUserPassword>
```

4.2 Create Signing Key

Create a signing key that will be used by the organization for e.g., code signing, or issue signing / sealing of documents. The key shall be generated inside the HSM and must be flagged as sensitive, and non-extractable.

```
java -jar keytoolX.jar -host <HsmUrlorIpAddr> -port <Port> -user
<PartitionName> -password <HsmUserPassword> -genkeypair -alias
<SigningKey> [-keypass <arg>] -keyalg <keyalg> -keysize <keysize>
-sigalg <sigalg> -dname CN=<SigningCertificateName> -validity <valDays>
```

Verify the key flags of the created key pair by using `primus-tools`.

```
java -jar primus-tools.jar ListKeyFlags -host <HsmUrlorIpAddr> -port
<Port> -user <HsmUserPassword> -password <HsmUserPassword> -keyname
<SigningKey>

Example response:
SigningKeyName sensitive nonextractable modifiable noncopyable token
nonindestructible private nonpublic neverextractable alwayssensitive
local decrypt sign unwrap notderive notintegrity notsignrecover
```

Moreover, one might want to use keytoolX to verify additional information.

```
java -jar keytoolX.jar -host <HsmUrlorIpAddr> -port <Port> -user
<PartitionName> -password <HsmUserPassword> -list -v -alias <SigningKey>

Example response:
Alias name: SigningKey
Creation date: 26.01.2022
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=SigningCertificateName
Issuer: CN=SigningCertificateName
Serial number: 7eddc529
Valid from: Wed Jan 26 15:48:42 CET 2022 until: Thu Jan 26 15:48:42 CET
2023
Certificate fingerprints:
      MD5:  EA:F1:FC:54:3C:B1:2D:BA:F7:9F:24:F3:11:9F:E8:6C
      SHA1: A7:39:1E:D9:17:90:56:5C:7D:64:6A:CE:76:6B:92:C5:FC:8E:95:BD
      SHA256:
0C:97:8A:86:11:08:A7:CF:B7:DF:67:17:E0:B6:F7:EA:15:F3:CA:19:54:C5:7B:80:
E0:14:0A:3D:AA:84:8C:5E
      Signature algorithm name: SHA256withRSA
      Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: B1 A5 7A 45 2C 73 89 2C   17 B9 6A E4 70 50 25 7C
..zE,s.,..j.pP%.
0010: 3D FE 25 90                                        =.%.
]
]
```

4.3 Generate CSR

Create the Certificate Signing Request (CSR) for the signing key. Again, we will use keytoolX to create the CSR, but one can also use their own application of choice here. keytoolX, creates the CSR in the working directory for the given filename.

```
java -jar keytoolX.jar -host <HsmUrlorIpAddr> -port <Port> -user
<PartitionName> -password <HsmUserPassword> -certreq -alias <SigningKey>
-keypass <arg> -file <SigningKey.csr>
```

4.4 Generate the Key Attestation File

Generate the Key Attestation File for the key (in our example the signing key we created with the help of keytoolX in chapter 4.2) you want to attest with the Key Attestation key.

```
java -jar primus-tools.jar GetAttestation -host <HsmUrlorIpAddr> -port
<port> -user <PartitionName> -password <HsmUserPassword> -keyname
<SigningKey> [-keypassword <KeyPassword>] -attestationkeyname
<KeyAttestationKey> -ca <Primus-Root-Certificate.crt> -out
<SigningKeyAttestationFile>
```

The command generates the following 4 files:

- SigningKeyAttestationFile.xml
- SigningKeyAttestationFile.crt
- SigningKeyAttestationFile.sig
- SigningKeyAttestationFile.sig.b64

The Key Attestation File (.xml) shows the key name, algorithm type and key size, creation date, the corresponding public key, the key attributes, followed by the attestation certificate chain (.crt) that is base64 encoded and contains the key attestation, device, and root certificate, and finally the signature of the Key Attestation File in binary (.sig) and Base64 (.sig.b64) encoding.

The splitting into different files and the provisioning of the signature file in binary format simplifies the verification procedures based on the OpenSSL software library.

Examples of the attestation files are provided in the annex, section 8.1.1 Attestation of Key (sample file).

4.5 Submit CSR and Key Attestation File

Now you can submit the CSR together with the Key Attestation File to the Qualified Trust Service Provider for validation and issuance of the certificate for the signing key.

5 Validation of the Key Attestation File and CSR by the QTSP

The Qualified Trust Service Provider validates the CSR and the Key Attestation File by performing the following checks, prior issuing the certificate:

- The Key Attestation File can be verified back to the Securosys Primus Root Certificate, which involves proofing the authenticity of:
  - the certificate chain, and
  - the signature
- The Key Attestation File belongs to the signing key from the CSR
- The signing key has the correct flags, e.g.: "`extractable=false`" and "`never_extractable=true`"

For highest assurance and to guarantee that all these checks can be conducted independently from any Securosys tools, we'll use the OpenSSL software library to perform these verifications.

Documentation of all OpenSSL command options can be found on the OpenSSL Documentation Page [2].

5.1 Verify Authenticity of Key Attestation File

Verify that the Key Attestation File is authentic and was generated by a Primus HSM with the following command. The option '`-untrusted`' is used to include the intermediate device certificate to construct a certificate chain from the subject certificate to the trust-anchor e.g., `Primus-Root-Certificate.crt`.

```
openssl verify -CAfile <Primus-Root-Certificate.crt> -untrusted
<SigningKeyAttestationFile>.crt <SigningKeyAttestationFile>.crt

Example of positive response:
<SigningKeyAttestationFile>.crt: OK

Example of negative response:
<SigningKeyAttestationFile>.certificatechain: C = CH, ST = ZH, L =
Zurich, O = Securosys, CN = SECUROSYS_ROOTKEY_DEV
error 19 at 2 depth lookup:self signed certificate in certificate chain
```

By the response "OK" you know that the certificate chain was issued by Primus HSM.

5.2 Verify the Signature of the Key Attestation File

Next you want to verify that the signature of the Key Attestation File is based on the key of the Key Attestation certificate in the certificate chain, which was verified to be issued by the Primus HSM before.

As an intermediate step, we'll extract the certificate's subject public key info block of the attestation key certificate and output it into a file in PEM format (`<SigningKeyAttestationFilePubKey.pem>`). This preparation step is needed to do further processing with the OpenSSL software library.

```
openssl x509 -pubkey -noout -in <SigningKeyAttestationFile>.crt -out
<SigningKeyAttestationFilePubKey.pem>
```

Now, verify the SHA256 signature using the public key extracted before from the attestation certificate chain from the message digest of the <SigningKeyAttestationFile>.

```
openssl dgst -verify <SigningKeyAttestationFilePubKey.pem> -sha256
-signature <SigningKeyAttestationFile>.sig
<SigningKeyAttestationFile>.xml

Example of positive response:
Verified OK

Example of negative response:
Verified Failure
```

With the response "Verified OK", there is evidence that the Key Attestation File, signature, and certificate chain belong together and are authentic.

5.3 Verify that the Key Attestation belongs to the CSR

For the verification that the Key Attestation File and the CSR belong together, different intermediate steps must be performed. This allows further processing using the OpenSSL software library.

First, extract the public key part of the certificate signing request file. The output file has the PEM format.

```
openssl req -in <SigningKeyCsr> -noout -pubkey -out
<SigningKeyCsrPubKey>.pem
```

Second, convert this public key from PEM to DER format so it can be compared later.

```
openssl asn1parse -noout -in <SigningKeyCsrPubKey>.pem -out
<SigningKeyCsrPubKey>.der
```

Then, extract the public key from the Key Attestation File.

```
grep public_key <SigningKeyAttestationFile>.xml | sed -e 's,[^>]*>,,' -e
's,<.*,,' | base64 -d > <SigningKeyAttestationFilePubKey>.der
```

Now, compare the two public keys.

```
cmp <SigningKeyCsrPubKey>.der <SigningKeyAttestationFilePubKey>.der &&
echo Verified OK

Example of positive response:
Verified OK

Example of negative response:
Any other response
```

With the response "Verified OK" there is proof that the CSR and the Key Attestation File match with each other.

5.4 Verify Key Attributes

After the authenticity of the Key Attestation File has been verified, it is time to verify if the signing key fulfills the requirements imposed by the regulation.

The evidence that the signing key was generated on the HSM and was never exported is provided by the key flag **"never_extractable=true".** This flag is set by the HSM automatically at key creation and indicates that the key was never outside of the HSM, and thus must have been generated inside the HSM. The flag **"extractable=false"** proves that this key can never be exported from the HSM.

Open the key attestation File, e.g., `<SigningKeyAttestationFile>.xml` and verify both flags:

- `extractable:` `false`
- `never_extractable:` `true`

Any XML parser may help you in reading the attestation file.

5.5 Issuance of certificate by QTSP

If all verification steps are positive, there is proof that the signing key, referred to in the certificate signing request file, is securely generated inside a Primus HSM and that it can never be exported from the HSM.

Depending on the requirements as set by the QTSP, the QTSP can now proceed with their certificate issuance process or request additional proof of the actual HSM configuration in which the signing key has been created. In that case, the organization needs to perform a device attestation and provide evidence of the actual HSM configuration, state, and logs to the QTSP/auditor for verification prior issuing the signing certificate. Device attestation (audit) and the verification of the audit data is described in the subsequent chapters.

5.6 Import the Signing Certificate

When the organization receives the signing certificate from the QTSP, they can start using it with their application of choice and start issuing signatures accordingly.

In our example, where `keytoolX` is used to create the signing key, `keyToolX` can be used as well to import the signing certificate in the HSM.

```
java -jar keytoolX.jar -host <HsmUrlorIpAddr> -port <Port> -user
<PartitionName> -password <HsmUserPassword> -importcert -alias <alias>
[-keypass <arg>] -file <filename>
```

6 Device Attestation (Audit)

As from Primus HSM firmware version 2.10 onwards, attestations on device level are available. These attestations may be issued by either 1) the SO role using either the CLI or Decanus Terminal, or 2) the Partition SO role for CloudsHSM users using the Partition Decanus Terminal and are exported as XML files onto a USB stick.

The attestation files might provide further evidence on the complete HSM environment. As the Device Attestation is based on the common device key, the Key Attestation can be clearly linked with the HSM environment.

All audit commands are accessible via:

- GUI below the AUDIT menu branch
- Console `hsm_audit_*` (see `helpaudit`)

Each audit export consists of an archive (zip-file) containing:

Filename	Description
<deviceName>.seal or *<deviceName>.sconfig* or *<deviceName>.state* or **.log*	Content file(s), e.g., digital seal, security configuration, device state or logs, timestamped and encoded as XML file or plain text file (logs only). sys.log, crypto.log, crash.log and partition logs (username.log).
**.sig*	Signature of the content file(s) using SHA256 digest
audit.cert	Certificate chain PEM encoded, containing audit, device, and Securosys root certificates

Table 3 Content of the audit export archive file

Note, that the content files are only signed if the root key store and audit key are initialized (see chapter 3.1).

The following sections are an extract of Primus HSM User Guide [1], and provide an overview of all relevant audit commands.

For CloudsHSM customers that don't have partition administration rights, the Securosys HSM operation team is able to provide the QTSP/auditor evidence of the HSM environment. Please contact the team via the Securosys Support Portal as described in chapter 1.4.

6.1 Attestation of Digital Seal

The following command writes the digital seal information together with current device timestamp into an XML file named `<deviceName>.seal`, signs the XML file (`*.sig`), and packs them together with the certificate chain (`audit.cert`) into an archive named `<deviceName>.seal.zip` and exports it onto a USB drive.

AUDIT -> EXPORT AUDIT -> SEAL

`hsm_audit_seal`

Chapter 8.1.2 shows an example of such XML seal file (`*.seal`) content.

6.2 Attestation of Device Security Configuration

Writes the relevant Device Security Configuration together with current device timestamp into an XML file named `<deviceName>.sconfig`, signs the XML file (`*.sig`), and packs them together with the certificate chain (`audit.cert`) into an archive named `<deviceName>.sconfig.zip`.

AUDIT -> EXPORT AUDIT -> CONFIGURATION 

`hsm_audit_config`

Chapter 8.1.3 shows an example of such XML security configuration file (`*.sconfig`) content.

6.3 Attestation of Device State

Writes the relevant Device Status Data together with current device timestamp into an XML file named `<deviceName>.state`, signs the XML file (`*.sig`), and packs them together with the certificate chain (`audit.cert`) into an archive named `<deviceName>.state.zip`.

AUDIT -> EXPORT AUDIT -> STATE 

`hsm_audit_state`

Chapter 8.1.4 shows an example of such XML device state file (`*.state`) content.

6.4 Attestation of Audit Logs

Writes all the log files (`sys.log, crypto.log, crash.log`) and partition log files (`<username>.log`) as separate plain text files (`*.log`), signs all these log files (`*.sig`), and packs them together with the certificate chain (`audit.cert`) into an archive named `<deviceName>.log.zip`.

AUDIT -> EXPORT AUDIT -> LOGS 

`hsm_audit_logs`

7 Verification of Signed Audit Information

Prerequisites for verification of the audit information are:

- Attestation file, signature file and certificate chain (contained in the `*.zip` archives)
- Securosys Root Certificate, e.g., `rsa_root_dev.crt` (see section 3.2)
- LibreSSL, OpenSSL or equivalent software library

The certificate chain (`audit.cert`) contains 3 certificates chained, in PEM encoded format:

1) Certificate of the device audit key with serial number, signed by intermediate device key
2) Certificate of the intermediate device key with serial number, signed by Securosys root key
3) Certificate of the Securosys root key (to be compared with the one downloadable from the Securosys Support portal)

The chained certificates can be viewed in detail using the following OpenSSL command:

```
openssl crl2pkcs7 -nocrl -certfile audit.cert | openssl pkcs7
-print_certs -text
```

The example below shows an excerpt of the subject and issuer of the chained certificates:

```
issuer=C = CH, ST = ZH, L = Zurich, O = Securosys, CN = PRIMUS HSM KEY (SN: 18376142)
subject=C = CH, ST = ZH, L = Zurich, O = Securosys, CN = PRIMUS HSM AUDIT KEY (SN: 18376142)

issuer=C = CH, ST = ZH, L = Zurich, O = Securosys, CN = PRIMUS_HSM_ROOTKEY
subject=C = CH, ST = ZH, L = Zurich, O = Securosys, CN = PRIMUS HSM KEY (SN: 18376142)

issuer=C = CH, ST = ZH, L = Zurich, O = Securosys, CN = PRIMUS_HSM_ROOTKEY
subject=C = CH, ST = ZH, L = Zurich, O = Securosys, CN = PRIMUS_HSM_ROOTKEY
```

7.1 Verification of the Audit Certificate Chain

Verify that the audit certificate chain (`audit.cert`) is authentic and was generated by a Primus HSM with the following command. The option '`-untrusted`' is used to include the intermediate device certificate to construct a certificate chain from the audit certificate to the trust-anchor e.g., `Primus-Root-Certificate.crt`.

```
openssl verify -CAfile <Primus-Root-Certificate.crt> -untrusted
audit.cert audit.cert

Example of positive response:
audit.cert: OK

Example of negative response:
audit.cert: C = CH, ST = ZH, L = Zurich, O = Securosys, CN =
SECUROSYS_ROOTKEY_DEV
error 19 at 2 depth lookup:self signed certificate in certificate chain
```

By the response "OK" you know that the certificate chain was issued by Primus HSM.

7.2 Verification of Signed Content/Log File

Next, verify that the signature of a content/log file is based on the key of the device audit certificate in the certificate chain, which was verified to be issued by the Primus HSM before.

Extract an audit archive into a folder e.g., HSM-PRIMUS-X-142.sconfig.zip, having:

- The certificate chain file (i.e., audit.cert)
- The content file (e.g., HSM-PRIMUS-X-142.sconfig)
- The signature file (e.g., HSM-PRIMUS-X-142.sconfig.sig)

As an intermediate step, we'll extract the certificate's subject public key info block of the device audit certificate (audit.cert) and output it into a file (audit.pubkey) in PEM format.

```
openssl x509 -pubkey -in audit.cert -noout -out audit.pubkey
```

Now, use the extracted public key to verify the SHA256 signature of a content file (e.g., HSM-PRIMUS-X-142.sconfig).

```
openssl dgst -sha256 -verify audit.pubkey -signature HSM-PRIMUS-X-
142.sconfig.sig HSM-PRIMUS-X-142.sconfig

Example of positive response:
Verified OK

Example of negative response:
Verified Failure
```

With the response "Verified OK", there is evidence that the content/log file and audit certificate chain belong together and are authentic.

8 Appendix

8.1 XML Reference Attestation (Audit) Files

8.1.1 Attestation of Key (sample file)

Example of a Key Attestation File for a key named "CompanySealKey03".

8.1.1.1 Key Attestation File (.xml)

```
<private_key creation="generated">
      <label>CompanySealKey03</label>
      <create_time>2022-01-26T14:48:44Z</create_time>
      <attest_time>2022-01-26T15:31:46Z</attest_time>
      <algorithm>RSA</algorithm>
      <algorithm_oid>1.2.840.113549.1.1.1</algorithm_oid>
      <key_size>2048</key_size>
      <public_key>MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAue8SnBk1Neye+dL2fA8v1Hzr3p
1OXBqpcnaQ0ivYreKyk5iUK3NUPvN849thansQxVy5tbTVqKeEWp/YohgzQ9EIlYmOdrkOZ8Mu2Gg1n17lC3JjHB
KcpHCQRcZEZYOHf/slPrgl0qwnLplWhOL238HbehkNyM6FVNWsX2KYh7uPDEsW/zvxrHJqN3KCn6DbmcD5ShPpBP
uIbWDUv80G5sAmKBaFfWenJIvFIR4BDa30CZCGyhfC3lHn83mNFfvH+wU8sPKtOg4yvlNA6kjhGLk7H8HRrEyRoV
rDtvgNxwqy46i1ZayV2OlHv/nIBUG95XsQTHqZVHHTQeMf1FFHHQIDAQAB</public_key>
      <attributes>
            <decrypt>true</decrypt>
            <sign>true</sign>
            <unwrap>true</unwrap>
            <derive>false</derive>
            <sensitive>true</sensitive>
            <always_sensitive>true</always_sensitive>
            <extractable>false</extractable>
            <never_extractable>true</never_extractable>
            <modifiable>true</modifiable>
            <copyable>false</copyable>
            <destroyable>true</destroyable>
            <unique>true</unique>
      </attributes>
</private_key>
```

8.1.1.2 Key Attestation Certificate Chain (.crt)

```
-----BEGIN CERTIFICATE-----
MIIDcTCCAlmgAwIBAgIRANFmmfL774ItZBYDr+37T9kwDQYJKoZIhvcNAQELBQAw
ZzELMAkGA1UEBhMCQ0gxCzAJBgNVBAgTAlpIMQ8wDQYDVQQHEwZadXJpY2gxEjAQ
BgNVBAoTCVNlY3Vyb3N5czEmMCQGA1UEAxMdUFJJTVVTIEhTTSBLRVkgKFNOOiAx
ODM3NjExMikwIBcNMjIwMTI2MTQ0MDI4WhgPMjA1MDA1MjExMTU3NTlaMF0xCzAJ
BgNVBAYTAkNIMQswCQYDVQQIEwJaSDEPMA0GA1UEBxMGWnVyaWNoMRIwEAYDVQQK
EwlTZWN1cm9zeXMxHDAaBgNVBAMTE0tleUF0dGVzdGF0aW9uS2V5MDMwggEiMA0G
CSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDPrXNF5svgxl8xvUlHUBCPCltMv2MC
WweCdtBnfCwOLfu7cXIkwIBfI1qjRCl94Uul8dHi5vjCZd5+AcKvH/b6SxkPBsKU
cwQ4+IDd0UQxSiOY71egJx/7JyT/Rr21baoN4tgXlft7Q+Gcw2KTGrW1sollqKc9
/Db75Qinhd2p8sRR7FOT5IO3MmH+3AHl5gC2u+d5ExiAW6OjlISFu/qMN7bwxnFN
L9ePTiFKessPxTAx5bzWf/43rupZHlEugJIMjDG30b4FkwxCq3whTZrS8nche8Wf
vT9paQy4KAcGQuvsTpXLDbeRuLyfbJIKWhHGng68+NPE/nrcUYKQv6b1AgMBAAGj
IDAeMAwGA1UdEwEB/wQCMAAwDgYDVR0PAQH/BAQDAgbAMA0GCSqGSIb3DQEBCwUA
A4IBAQBcVv1G28mcgRG7ueznPAB7KoZ4dYHsqfsIY5U5UJ7Kwgx2/5spSi80WF/s
UedCGXHJwklb8iCIEqb+rx8AIby/Xyu0BQBazNCmGqRVVg8A6gANgYYQkELwIJGF
pYALK5tXOVhOKJtoxCO9bT3clepo99EyfXOFIJgkj53NaHbVDzj8aV2e7e7tTnLC
slqSfOhBN8V+R09/9d+S5kFdhMTy1ZBHaQJ77lhPr2QwqRMTc1QvSYrBMAQu3p1M
ZbL+yHBXAbQBy52ylen8Vu+FYKcqTKayAziJGHx2HY95U4jwYEzxzyfd5zzBVTxd
H/6JQ20kBfP4uHp32iyV5hm8XErs
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
```

```
MIIDszCCApugAwIBAgIRAOIL67rqTTHrSGbLcANgQ+4wDQYJKoZIhvcNAQELBQAw
XzELMAkGA1UEBhMCQ0gxCzAJBgNVBAgTAlpIMQ8wDQYDVQQHEwZadXJpY2gxEjAQ
BgNVBAoTCVNlY3Vyb3N5czEeMBwGA1UEAxQVU0VDVVJPU1lTX1JPT1RLRVlfREVW
MCAXDTIxMDkwODExMjU1MVoYDzIwNTAwNTIxMTE1NzU5WjBnMQswCQYDVQQGEwJD
SDELMAkGA1UECBMCWkgxDzANBgNVBAcTBlp1cmljaDESMBAGA1UEChMJU2VjdXJv
c3lzMSYwJAYDVQQDEx1QUklNVVMgSFNNIEtFWSAoU046IDE4Mzc2MTEyKTCCASIw
DQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAIMFVGCdz1xgVwLU8xY1k4ngSMXq
nzEaEj1FyGx4OITs29pKMw/eg/8QKvDaw/7BlicfU9p2IhuMI0Q1qH2uJmsSiHm/
s6hjJkZCpe4p/YM+vIeXli6OOP83oPkjkvV01i4cnnm2s83UfT3em89Bzw2JLWtE
39zbEyi/qB/xwZxkrrdIiaW4bceBCweE8iXJ+8mf1a9q/EYIWJPT+o2pO57XhEhc
HUf9PuuXYy9jY03C/lYVmBAUXG9hhYO33SsEH6+YiO04Sar5NYZQRG8BADY7ThGZ
fOPcVe3aWt3btxMLcxahgc31nswiXpa18YLzxYDOxV6F7Kp5Trq95XqXsUECAwEA
AaNgMF4wEgYDVR0TAQH/BAgwBgEB/wIBADA4BgNVHSAEMTAvMC0GCysGAQQBgtx8
BAEBMB4wHAYIKwYBBQUHAgEWEHd3dy5zZWN1cm9zeXMuY2gwDgYDVR0PAQH/BAQD
AgFGMA0GCSqGSIb3DQEBCwUAA4IBAQAkMhsXi+A6UOamNzVbemdMAY1eOdZzEScd
6iLZQI0OvOh8IKNYAODIzm7HKNuaqlL/9G9BQPu2MZIzf8MffxudNJzYB6uBFNzx
T81kHBBhoDYxZr2Lj0r30jBPLF4CcnpNB/S2FinqhWRvG5Yub9JqMPKr/V+f+UIz
GTio96gygYWGQQSLhwcn2llDnVspRnB9eTvdckdHL6CXbnfPeYxS0JFjPEF4FV9s
0KZz62TP0XKq1hgKPAJtKvAUrtcRRBLJm2xTnsfelV2LoY5yGRJk5gHr0YIDJuoZ
IDyhE1LKs2fnl7tcwEkt6gABizTPevs29hiC8PlX3q4UEPXfyoxz
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIDqzCCApOgAwIBAgIRAJ2TU2SzRdjLZtI48GsWaNEwDQYJKoZIhvcNAQELBQAw
XzELMAkGA1UEBhMCQ0gxCzAJBgNVBAgTAlpIMQ8wDQYDVQQHEwZadXJpY2gxEjAQ
BgNVBAoTCVNlY3Vyb3N5czEeMBwGA1UEAxQVU0VDVVJPU1lTX1JPT1RLRVlfREVW
MCAXDTIwMDUyODExNTc1OVoYDzIwNTAwNTIxMTE1NzU5WjBfMQswCQYDVQQGEwJD
SDELMAkGA1UECBMCWkgxDzANBgNVBAcTBlp1cmljaDESMBAGA1UEChMJU2VjdXJv
c3lzMR4wHAYDVQQDFBVTRUNVUk9TWVNfUk9PVEtFWV9ERVYwggEiMA0GCSqGSIb3
DQEBAQUAA4IBDwAwggEKAoIBAQCmLFqDEqpNXtPjFJtRIwscC0uwUZO1sdWNZp61
WI1hiuVNpT2HNSOVg1AhHWbxbp5UaDvExRbC3jYboi1ybVWHRatfyT6cFuOP8hkk
qMnhr62C5btLDqu9/KOfM6p7RE7BijVpJgTJV+ZSuroB0hLz5a7CMGEJq/UYFbqe
sUQX7I73w732fZIxhtPSciNgmhIrAzTiuyz+wkMcequatZxPi5CaoGeFgZxDxXxp
0XT7bSatWuuyPtz9oyVchvqZGiC8wS62xl1ehrJTLbq9XHA2LOK35ax68hzNzowP
N96ytpyq91G8zSA8l8MUps2qLkKmZW6sCCvKThTe49cfd331AgMBAAGjYDBeMBIG
A1UdEwEB/wQIMAYBAf8CAQEwDgYDVR0PAQH/BAQDAgFGMDgGA1UdIAQxMC8wLQYL
KwYBBAGC3HwEAQEwHjAcBggrBgEFBQcCARYQd3d3LnNlY3Vyb3N5cy5jaDANBgkq
hkiG9w0BAQsFAAOCAQEANfnMu0WqGsh6NmgXBJamQB243UcBXxJoJzkdEB9Icwyj
Eb9c1KwA0TXDuwHpyhlPTFKsPHEhg+dRNLsO0NZIQ1ITQE3WI5P1zwEYRm3pSLHD
KBFIielk5GC3zYMQ3GoSU6VRdfDob65Jol3k3Z/WV/KV7YM+GQJPHPL/+4xYK166
L0SKGg6wJwipyo77fErxvbdOuJwDsjWqDIyrd7VLz6LhjRoxN2mMWcIvB2J0cGAx
wc3n48a+9NGGIOFnCfTl7ABniQSGu7iYU50bisVrvypLdd2q+q64nPNZiMkE/ruy
dw7tiW74QqcYcnMFjEKVM7OQBSz3bt7c7Zu/tIwV4g==
-----END CERTIFICATE-----
```

8.1.1.3 Key Attestation Signature (.sig.b64)

The key attestation signature is in binary format for better processing with OpenSSL, below encoded to Base64 for better reading.

```
E3YXV5SkosphRSRjGO9NK0tWnhjT6dqscZ4FkFDsRTg7pXSh8PLuc57appRY2oMCdfu7zL2ZqTaUTi7a74eYse/A
qieLKNpJrWNBC5InKhje1k/rn8zQVeyBzHuy8yrAK7sRBb/k6F9C1coBWjsy6bIdmB/WtykVL5YeQ2MwgG5uDd7O
eqw8eUiUzLXA2i2Pzd67A9g4VoV87KRct+wpDvhni95sa7zEvjU7iFwngtGPI7YmSYNyw/P4kd2Pd4eZa3k484hl
Yj2xfHoE93aEe/7lUFyFaIj0EPrX5+C9CUZWNPj8CYJdO/FpWzJBE8LBIufS3CBuN3aV7Io912/FAg==
```

8.1.2 Attestation of Digital Seal (.seal)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<HSM name="HSM-PRIMUS-X-142" state="master" serialnumber="18376142" version="2.10.0">
      <system_time>2021-10-06 07:03:41 UTC</system_time>
      <digital_seal>DS0HV-Q2POP-HXD</digital_seal>
</HSM>
```

8.1.3 Attestation of Device Security Configuration (.sconfig)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<primus version="2" minor="2" name="HSM-PRIMUS-X-142" state="master"
serialnumber="18376142">
      <system_time>2021-10-06 07:03:49 UTC</system_time>
      <syslog logserver="enabled" cryptoserver="enabled" gapless_auditlog="enabled">
            <syslog_level>6</syslog_level>
            <crypto_level>6</crypto_level>
            <syslog_address id="1"/>
            <syslog_size>5000</syslog_size>
            <syslog_port tcp="disabled">514</syslog_port>
            <syslog_interface>2</syslog_interface>
            <crypto_address id="1"/>
            <crypto_size>5000</crypto_size>
            <crypto_port tcp="disabled">514</crypto_port>
            <crypto_interface>2</crypto_interface>
      </syslog>
      <network>
            <resolv_config>
                  <domain_name/>
                  <nameserver_ipv4 id="1"/>
                  <nameserver_ipv6 id="1"/>
            </resolv_config>
            <data dhcp="disabled" dhcp6="disabled">
                  <autoNeg>enabled</autoNeg>
                  <lineSpeed>100</lineSpeed>
                  <fullDuplex>enabled</fullDuplex>
                  <ipv4addr>10.250.100.100</ipv4addr>
                  <ipv4mask>24</ipv4mask>
                  <ipv4gate>0.0.0.0</ipv4gate>
                  <ipv6addr>::</ipv6addr>
                  <ipv6mask>0</ipv6mask>
                  <ipv6gate>::</ipv6gate>
            </data>
            <management dhcp="disabled" dhcp6="disabled">
                  <autoNeg>enabled</autoNeg>
                  <lineSpeed>100</lineSpeed>
                  <fullDuplex>enabled</fullDuplex>
                  <ipv4addr>0.0.0.0</ipv4addr>
                  <ipv4mask>0</ipv4mask>
                  <ipv4gate>0.0.0.0</ipv4gate>
                  <ipv6addr>::</ipv6addr>
                  <ipv6mask>0</ipv6mask>
                  <ipv6gate>::</ipv6gate>
            </management>
            <high-availability dhcp="disabled" dhcp6="disabled">
                  <autoNeg>enabled</autoNeg>
                  <lineSpeed>100</lineSpeed>
                  <fullDuplex>enabled</fullDuplex>
                  <ipv4addr>0.0.0.0</ipv4addr>
                  <ipv4mask>0</ipv4mask>
                  <ipv4gate>0.0.0.0</ipv4gate>
                  <ipv6addr>::</ipv6addr>
                  <ipv6mask>0</ipv6mask>
                  <ipv6gate>::</ipv6gate>
            </high-availability>
            <redundancy dhcp="disabled" dhcp6="disabled">
                  <autoNeg>enabled</autoNeg>
                  <lineSpeed>100</lineSpeed>
                  <fullDuplex>enabled</fullDuplex>
                  <ipv4addr>0.0.0.0</ipv4addr>
                  <ipv4mask>0</ipv4mask>
                  <ipv4gate>0.0.0.0</ipv4gate>
                  <ipv6addr>::</ipv6addr>
                  <ipv6mask>0</ipv6mask>
                  <ipv6gate>::</ipv6gate>
            </redundancy>
```

```
        <static_ipv4_route id="1">
                <network>0.0.0.0</network>
                <mask>0</mask>
                <gateway>0.0.0.0</gateway>
                <interface>1</interface>
        </static_ipv4_route>
        <static_ipv6_route id="1">
                <network>::</network>
                <mask>0</mask>
                <gateway>::</gateway>
                <interface>1</interface>
        </static_ipv6_route>
</network>
<config_process>
        <single_port>enabled</single_port>
        <redundancy>1</redundancy>
        <bonding>disabled</bonding>
        <snmp_interface>2</snmp_interface>
        <password/>
        <snmp>enabled</snmp>
        <backup>enabled</backup>
        <restore>enabled</restore>
        <local_clone>enabled</local_clone>
        <remote_update>disabled</remote_update>
        <modify_config>enabled</modify_config>
        <add_genesis>enabled</add_genesis>
        <noise_source>1</noise_source>
        <tilt_sensor>enabled</tilt_sensor>
</config_process>
<jce_process>
        <active>enabled</active>
        <port>2300</port>
        <interface>1</interface>
</jce_process>
<pkcs_process>
        <active>enabled</active>
        <port>2310</port>
        <interface>1</interface>
</pkcs_process>
<mscng_process>
        <active>enabled</active>
        <port>2320</port>
        <interface>1</interface>
</mscng_process>
<ha_process>
        <active>disabled</active>
        <port>2330</port>
        <interface>3</interface>
</ha_process>
<decanus_process>
        <active>enabled</active>
        <port>2340</port>
        <interface>2</interface>
</decanus_process>
<crypto_process>
        <master_url id="1"/>
        <remote_ha_clone>disabled</remote_ha_clone>
        <max_user>20</max_user>
        <max_so>10</max_so>
        <delete_user>disabled</delete_user>
        <delete_last_user>disabled</delete_last_user>
        <crypto_access>enabled</crypto_access>
        <setup_password_lifespan>7300</setup_password_lifespan>
        <login_attempt_threshold>100</login_attempt_threshold>
        <attempt_reset_time>300</attempt_reset_time>
        <login_lockout_time>300</login_lockout_time>
        <keystore_size_limit>0</keystore_size_limit>
```

```
            <import_keys>enabled</import_keys>
            <export_keys>enabled</export_keys>
            <extract_keys>disabled</extract_keys>
            <crypto_log>enabled</crypto_log>
            <clone_modify>enabled</clone_modify>
            <user_config>enabled</user_config>
            <invalidate_keys>disabled</invalidate_keys>
            <verify_block_state>enabled</verify_block_state>
            <session_objects>enabled</session_objects>
            <destroy_objects>enabled</destroy_objects>
            <use_objects>enabled</use_objects>
            <trust_store>disabled</trust_store>
            <pkcs_password/>
            <crypto_user state="disabled">
                  <user_name>DEMO-CAR</user_name>
                  <size_limit>0</size_limit>
                  <setup_password_lifespan>72</setup_password_lifespan>
                  <import_keys>disabled</import_keys>
                  <export_keys>disabled</export_keys>
                  <extract_keys>disabled</extract_keys>
                  <clone_modify>enabled</clone_modify>
                  <read_only>disabled</read_only>
                  <invalidate_keys>disabled</invalidate_keys>
                  <verify_block_state>enabled</verify_block_state>
                  <management>disabled</management>
                  <key_authorization>disabled</key_authorization>
                  <session_objects>enabled</session_objects>
                  <destroy_objects>enabled</destroy_objects>
                  <use_objects>enabled</use_objects>
                  <trust_store>disabled</trust_store>
                  <pkcs_password/>
                  <crypto_access>enabled</crypto_access>
                  <rest_api>disabled</rest_api>
                  <tsb_engine>disabled</tsb_engine>
                  <userlog>disabled</userlog>
                  <userlog_size>10240</userlog_size>
                  <userlog_level>6</userlog_level>
                  <jce_allowed>disabled</jce_allowed>
                  <pkcs_allowed>disabled</pkcs_allowed>
                  <mscng_allowed>disabled</mscng_allowed>
            </crypto_user>
            <crypto_user state="enabled">
                  <user_name>DEMO-CAS</user_name>
                  …
            </crypto_user>
            <locked_user state="enabled">
                  <user_name>REMOTE_MANAGED</user_name>
      …
            </locked_user>
      </crypto_process>
      <ntp state="enabled">
            <server id="1">ntp.securosys.ch</server>
            <interface>2</interface>
      </ntp>
</primus>
```

8.1.4 Attestation of Device State (*.state)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<HSM name="HSM-PRIMUS-X-142" state="master" serialnumber="18376142" version="2.10.0">
      <system_time>2021-10-06 07:03:55 UTC</system_time>
      <hardware>
            <authentication>card</authentication>
            <cpu>1.4.1594.2</cpu>
            <board>32.4.18.2</board>
            <integrity>OK</integrity>
```

```
        <temperature>
                <sensor name="Accelerator 0 core">46</sensor>
                <sensor name="Accelerator 1 core">47</sensor>
                <sensor name="Processor environment">36</sensor>
                <sensor name="Accelerator 0 environment">40</sensor>
                <sensor name="Accelerator 1 environment">36</sensor>
                <sensor name="Random generator environment">33</sensor>
                <sensor name="System environment">30</sensor>
                <sensor name="Processor ambient">34</sensor>
                <sensor name="Accelerator 0 ambient">39</sensor>
                <sensor name="Accelerator 1 ambient">35</sensor>
                <sensor name="Random generator ambient">31</sensor>
        </temperature>
        <cooling avg_rpm="123">
                <fan location="Left">122</fan>
                <fan location="Mid-left">124</fan>
                <fan location="Mid-right">125</fan>
                <fan location="Right">121</fan>
        </cooling>
        <cryptography>
                <fpga type="Accelerator EC">1.8.5.1.419824918</fpga>
                <fpga type="Accelerator RSA">1.3.8.1.419825170</fpga>
                <fpga type="Licensed Accelerator">4.16.16.4.0</fpga>
        </cryptography>
</hardware>
<firmware cryptographic_mode="Normal">
        <hsm>RX-2.10.0</hsm>
        <bootloader>V02.08.0000-rel</bootloader>
        <root_key_element>V1.0.0</root_key_element>
        <rollback>V1.0.0</rollback>
        <dmu state="protected">V2.8.43</dmu>
        <dmu_bootloader>V1.0.0</dmu_bootloader>
        <communication>
                <protocol name="jce">5.0.59</protocol>
                <protocol name="pkcs">5.0.57</protocol>
                <protocol name="mscng">5.0.57</protocol>
                <protocol name="ha">5.1.213</protocol>
                <protocol name="decanus">5.0.15</protocol>
                <protocol name="mgmt">5.0.3</protocol>
        </communication>
</firmware>
<license>
        <partitions>1000</partitions>
        <partition_size>150</partition_size>
        <partition_managers>1000</partition_managers>
        <ska>1000</ska>
        <rest_api>1000</rest_api>
        <tsb>1000</tsb>
        <upper_version>v2</upper_version>
        <lower_version>v2</lower_version>
        <module name="Device backup"/>
        <module name="Remote update"/>
        <module name="High availability"/>
        <module name="Device cloning"/>
        <module name="Timestamp service"/>
        <module name="Decanus interface"/>
        <module name="Partition log"/>
        <module name="Root key store"/>
        <module name="JCE interface"/>
        <module name="MSCNG interface"/>
        <module name="PKCS interface"/>
…
</license>
<high_availability>N/A</high_availability>
<security_officer>
        <device>
                <device_security_officer id="0">SO01</device_security_officer>
```

```
                <device_security_officer id="1">SO02</device_security_officer>
        </device>
    </security_officer>
    <decanus>
        <decanus_terminal state="unknown">83193CDE9419175A</decanus_terminal>
    </decanus>
    <partition name="DEMO-CAR" state="open">
        <capacity>0.013</capacity>
        <size type="KB">20.0</size>
        <objects total="2">
            <remaining>511691</remaining>
            <private_key>1</private_key>
            <public_key>1</public_key>
            <symmetric_key>0</symmetric_key>
            <certificate>0</certificate>
            <data>0</data>
            <invalidated>0</invalidated>
        </objects>
        <setup_password state="Expired"/>
        <counter id="F39BCEFB-BEA8E0CA-6B471E54-C855E038">0</counter>
        <management>N/A</management>
    </partition>
    <partition name="DEMO-CAS" state="open">
        …
    </partition>
…
```

8.2 Key Attestation JCE CLI Sample

The algorithms in the samples below are suggestions and can/should be changed.

Please reference the Appendix in the UserGuide to check for supported Algorithm -and Signatures.

8.2.1 Create a Key Attestation Key

Used KeyType is RSA, KeySize is 4096 and KeyName is key-attestation-key, adjust according to your needs

```
java -jar primus-tools.jar CreateAttestationKey -host a-api.cloudshsm.com -port 2300
      -user <PartitionName> -password <HsmUserPassword>
      -primusproxyuser <ProxyUser> -primusproxypassword <ProxyPassword>
      -keyname key-attestation-key -type RSA -size 4096
```

Check the returned list after command execution that the key key-attestation-key was generated successfully, look for key entry: ***key-attestation-key RSA 4096***

```
java -jar primus-tools.jar ListKeyStoreObjects -host a-api.cloudshsm.com -port 2300
      -user <PartitionName> -password <HsmUserPassword>
      -primusproxyuser <ProxyUser> -primusproxypassword <ProxyPassword>
```

8.2.2 Create a Signing Key

Create a signing key that will be used by the organization for e.g., code signing, or issue signing / sealing of documents. The key shall be generated inside the HSM and must be flagged as sensitive, and non-extractable.

Adjust the parameters keyalg*,* keysize*,* sigalg*,* dname *and* validity *according to your needs.*
The **validity** (in days, e.g., 3650 days = 10 years) *when generating the key is used for the certificate. If no validity parameter is specified, the certificate is valid for 3 months by default. However, this validity is not included in the CSR and is often overwritten by the CA policy anyway.*

```
java -jar keytoolX.jar -host a-api.cloudshsm.com -port 2300
      -user <PartitionName> -password <HsmUserPassword>
      -proxyuser <ProxyUser> -proxypassword <ProxyPassword>
      -genkeypair -alias SigningKey_Securosys -keyalg RSA -keysize 4096
      -sigalg SHA256withRSA -dname CN=Securosys -validity 3650
```

Check the returned key flags to the created key pair, the key should have the following flags: SigningKey_Securosys **sensitive nonextractable** modifiable noncopyable token nonindestructible private nonpublic neverextractable alwayssensitive local decrypt sign unwrap notderive notintegrity notsignrecover

```
java -jar primus-tools.jar ListKeyFlags -host a-api.cloudshsm.com -port 2300
      -user <PartitionName> -password <HsmUserPassword>
      -primusproxyuser <ProxyUser> -primusproxypassword <ProxyPassword>
      -keyname SigningKey_Securosys
```

8.2.3 Create the Certificate Signing Request (CSR)

This command creates the Certificate Signing Request for the signing key and will be saved in the working directory under the given filename "SigningKey_Securosys.csr".

Later you can submit the CSR to the Qualified Trust Service Provider (QTSP) for validation and issuance for the certificate for the signing key.

```
java -jar keytoolX.jar  -host a-api.cloudshsm.com -port 2300
      -user <PartitionName> -password <HsmUserPassword>
      -proxyuser <ProxyUser> -proxypassword <ProxyPassword>
      -certreq -alias SigningKey_Securosys -keypass 123456 -file
SigningKey_Securosys.csr
```

8.2.4 Generate Key Attestation File

This command creates the following Key Attestation Files. (You want to attest with the Key Attestation key):

- SigningKeyAttestationFile.xml
- SigningKeyAttestationFile.crt
- SigningKeyAttestationFile.sig
- SigningKeyAttestationFile.sig.b64

```
java -jar primus-tools.jar GetAttestation -host a-api.cloudshsm.com -port 2300
      -user <PartitionName> -password <HsmUserPassword>
      -primusproxyuser <ProxyUser> -primusproxypassword <ProxyPassword>
      -keyname SigningKey_Securosys -keypassword 123456
      -attestationkeyname key-attestation-key -ca hsm_rsa_root.crt
      -out SigningKeyAttestationFile
```

8.2.5 Submit CSR and Key Attestation File

Now you can submit the CSR together with the Key Attestation File to the Qualified Trust Service Provider for validation and issuance of the certificate for the signing key.

8.3 Key Attestation JCE API Code Sample

Key attestation can also be performed by using JCE API directly. The following Java sample illustrates how to retrieve a key attestation from a Securosys Primus HSM and verify its signature and certificate chain.

```
/*
 * Copyright (C) 2019,2021, Securosys SA
 */

import java.io.PrintStream;
import java.nio.charset.StandardCharsets;
import java.security.Security;
import java.security.Signature;
import java.security.cert.CertPath;
import java.security.cert.CertPathValidator;
import java.security.cert.Certificate;
import java.security.cert.CertificateFactory;
import java.security.cert.PKIXParameters;
import java.security.cert.TrustAnchor;
import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import com.securosys.primus.jce.PrimusAttestation;
import com.securosys.primus.jce.PrimusConfiguration;
import com.securosys.primus.jce.PrimusEncoding;
import com.securosys.primus.jce.PrimusLogin;
import com.securosys.primus.jce.PrimusProvider;
import com.securosys.primus.jce.PrimusProviderException;
import com.securosys.primus.jce.file.FileContents;

/**
 * Sample how to retrieve a key attestation
 * from a Securosys Primus HSM,
 * and verify its signature and certificate chain.
 */
public class KeyAttestation {

      public static void main(final String... args) throws Exception {

             // output file; see '-out' for file output
             PrintStream out = System.out;

             // HSM name and credentials
             String host = null;
             String port = "2300";
             String user = null;
             char[] password = null;

             // key names and credentials
             String attestationkeyname = null;
             char[] attestationkeypassword = null;
             String keyname = null;
             char[] keypassword = null;

             // CA PEM file
             String fileca = null;

             // output files base
             String fileout = null;

             for (int i = 0; i < args.length; i++) {
                    final String arg = args[i];
                    final String nextarg = (i < args.length - 1 ? args[i + 1] : null);
```

```
                        if (arg.equals("-host")) {
                                host = nextarg;
                                i++;
                        } else if (arg.equals("-port")) {
                                port = nextarg;
                                i++;
                        } else if (arg.equals("-user")) {
                                user = nextarg;
                                i++;
                        } else if (arg.equals("-password")) {
                                password = (nextarg == null ? null : nextarg.toCharArray());
                                i++;
                        } else if (arg.equals("-attestationkeyname")) {
                                attestationkeyname = nextarg;
                                i++;
                        } else if (arg.equals("-attestationkeypassword")) {
                                attestationkeypassword = (nextarg == null ? null
nextarg.toCharArray());
                                i++;
                        } else if (arg.equals("-keyname")) {
                                keyname = nextarg;
                                i++;
                        } else if (arg.equals("-keypassword")) {
                                keypassword = (nextarg == null ? null :
nextarg.toCharArray());
                                i++;
                        } else if (arg.equals("-ca")) {
                                fileca = nextarg;
                                i++;
                        } else if (arg.equals("-out")) {
                                fileout = nextarg;
                                i++;
                        }
                }

                // install/configure/login
                Security.addProvider(new PrimusProvider());
                PrimusConfiguration.setHsmHost(host, port, user);
                PrimusLogin.login(user, password);

                // holders for signature and chain
                final byte[][] signature_ = new byte[1][];
                final String[] chain_ = new String[1];

                // get the attestation XML, together with a signature of it, and the
certificate chain used for the signature
                final String xml =
PrimusAttestation.getSignedAttributes(attestationkeyname, attestationkeypassword,
keyname, keypassword, signature_, chain_);

                final byte[] signature = signature_[0];
                final String chain = chain_[0];

                // print data
                if (fileout == null) {
                        out.println(xml);
                        out.println(PrimusEncoding.base64Encode(signature));
                        out.println();
                        out.println(chain);
                } else {
                        FileContents.putContents(fileout, xml);
                        FileContents.putContents(fileout + ".signature", signature);
                        FileContents.putContents(fileout + ".certificatechain", chain);
                }

                // decode certificate chain: from list of PEM encoded certificates
                final Certificate[] cc = PrimusAttestation.decodeCertificateChain(chain);
```

```
            final Certificate c = cc[0];
            final Certificate ca = cc[cc.length - 1];
            // let check the chain: this only verifies signatures, not expiry nor X.509
flags
            PrimusAttestation.decodeCertificateChain(chain, ca);

            // check the chain against provided CA
            if (fileca != null) {
                  PrimusAttestation.decodeCertificateChain(
                        chain,
                        PrimusAttestation.decodeCertificateChain(
                              FileContents.getContentsAsString(fileca))[0]);
            }

            // check X.509 values, such as expiry
            final TrustAnchor ta = new TrustAnchor((X509Certificate)ca, null);
            final Set<TrustAnchor> tas = new HashSet<TrustAnchor>();
            tas.add(ta);
            final List<Certificate> l = new ArrayList<Certificate>();
            for (final Certificate ccc : cc) {
                  l.add(ccc);
            }
            final CertPath p =
CertificateFactory.getInstance("X.509").generateCertPath(l);
            final PKIXParameters pp = new PKIXParameters(tas);
            pp.setRevocationEnabled(false); // needed to not get the complaint about
missing OCSP URL (java.security.cert.CertPathValidatorException: Could not determine
revocation status)

      CertPathValidator.getInstance(CertPathValidator.getDefaultType()).validate(p, pp);
// may throw GeneralSecurityException
            out.println("certificate path validated");

            // verify attestation data signature
            String signAlgorithm = PrimusAttestation.getRsaSignAlgorithm();
            try { // derLength requires primusX.jar 2.2.3 or newer
                  if (PrimusEncoding.derLength(signature) > 0) { // RSA signature is
no DER, EC is
                        signAlgorithm = PrimusAttestation.getEcSignAlgorithm();
                  }
            } catch (final NoSuchMethodError e) {
                  //
            }
            final Signature verifyer = Signature.getInstance(signAlgorithm);
            verifyer.initVerify(c);
            verifyer.update(xml.getBytes(StandardCharsets.UTF_8));
            final boolean verified = verifyer.verify(signature);

            if (verified) {
                  out.println("signature verified");
            } else {
                  throw new PrimusProviderException("signature verification failed");
            }
      }
}
```

8.4 Key Attestation REST API Sample

Key attestation can also be performed by using the REST API. In the following subsections, an example is given of all relevant REST API commands with regards to key attestation, e.g.:

- Retrieval of the certificate chain for the attestation key
- The creation of a key including key attestation
- Attestation of a key after its creation

8.4.1 GET/v1/attestation/certificate

Returns certificate of attestation key as a base64 encoded string.

```
curl -X 'GET' \
  'https://tsb-demo.cloudshsm.com/v1/attestation/certificate' \
  -H 'accept: application/json'

Example of a successful operation:
{
  "attestationCertificateChain": [

"MIIDbTCCAlWgAwIBAgIRALhq9mLNsoePWVAjAXloY9EwDQYJKoZIhvcNAQELBQAwZzELMAkGA1UEBhMCQ0gxCzA
JBgNVBAgTAlpIMQ8wDQYDVQQHEwZadXJpY2gxEjAQBgNVBAoTCVNlY3Vyb3N5czEmMCQGA1UEAxMdUFJJTVVTIEh
TTSBLRVkgKFNOOiAxODM3NjExMikwIBcNMjIwNTI0MTIwNDI1WhgPMjA1MDA1MjExMTU3NTlaMFkxCzAJBgNVBAY
TAkNIMQswCQYDVQQIEwJaSDEPMA0GA1UEBxMGWnVyaWNoMRIwEAYDVQQKEwlTZWN1cm9zeXMxGDAWBgNVBAMTD2F
0dGVzdGF0aW9uLWtleTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANg1AsyPyWVTdk6gu6DCG7inbcw
mMj9q/Kin0UDN8X4hpSXtoWbRY9Saj9+oK3G92fFnmbB8VuxRYi6B/yRzwaSNi4PpBpesuDso6o/0Dvy0OPYRKFd
9W3+KsnxNDM1F/34FHwY/nKt0FcfRh71HflYO3qtFifW/D93tFZEJlxdlwl6x6f/zEkVxLK+3AJZXhELry5c27Vt
598/BGooVWMqLN8orDEoTNsbUR/Y0t3KpsJ9h3IATeJaFHtlSoonmvU98MlcPGbxPjkBYKOPCy7Fu0XZFpWDX22P
HJQFq90D569zcNt1OMVDTfYkTzS4XgaizPQmYX8jN+l2yUYKAlWUCAwEAAaMgMB4wDAYDVR0TAQH/BAIwADAOBgN
VHQ8BAf8EBAMCBsAwDQYJKoZIhvcNAQELBQADggEBAEKAAFrsTVjTAppXCrldztUhYP4Sggcy79dXdvqZSBlZIy/
78T4tMEJfcu46QJDe7qoC2clTFKhQNy+V1kyFQWFjcoDGvdqSWDQSAaqw61ZvNPaoHr9jIxFxQ66UgZO+jqjyHLS
bajgWhbDrDXTL5EIHWPJ/yjgKO43q97D4jF7cGq7j+hBX+rzFFNeAYAXsXJTJHftduJkV2gNQrnRTABAfL9C/Db9
YAlp5Dlp0Jrwtl4Uv2rVzp4Gk63MdgPIpVaacTfiMzPuQPgvkHTSVcCLpqUAHE7O+WIB+W0bLMMwE4GoSDZjR0PT
FRashr6Nj6ALGrCosStsh5TkQ/Zi8dvQ=",

"MIIDszCCApugAwIBAgIRAOIL67rqTTHrSGbLcANgQ+4wDQYJKoZIhvcNAQELBQAwXzELMAkGA1UEBhMCQ0gxCzA
JBgNVBAgTAlpIMQ8wDQYDVQQHEwZadXJpY2gxEjAQBgNVBAoTCVNlY3Vyb3N5czEeMBwGA1UEAxQVU0VDVVJPU1l
TX1JPT1RLRVlfREVWMCAXDTIxMDkwODExMjU1MVoYDzIwNTAwNTIxMTE1NzU5WjBnMQswCQYDVQQGEwJDSDELMAk
GA1UECBMCWkgxDzANBgNVBAcTBlp1cmljaDESMBAGA1UEChMJU2VjdXJvc3lzMSYwJAYDVQQDEx1QUklNVVMgSFN
NIEtFWSAoU046IDE4Mzc2MTEyKTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAIMFVGCdz1xgVwLU8xY
1k4ngSMXqnzEaEj1FyGx4OITs29pKMw/eg/8QKvDaw/7BlicfU9p2IhuMI0Q1qH2uJmsSiHm/s6hjJkZCpe4p/YM
+vIeXli6OOP83oPkjkvV01i4cnnm2s83UfT3em89Bzw2JLWtE39zbEyi/qB/xwZxkrrdIiaW4bceBCweE8iXJ+8m
f1a9q/EYIWJPT+o2pO57XhEhcHUf9PuuXYy9jY03C/lYVmBAUXG9hhYO33SsEH6+YiO04Sar5NYZQRG8BADY7ThG
ZfOPcVe3aWt3btxMLcxahgc31nswiXpa18YLzxYDOxV6F7Kp5Trq95XqXsUECAwEAAaNgMF4wEgYDVR0TAQH/BAg
wBgEB/wIBADA4BgNVHSAEMTAvMC0GCysGAQQBgtx8BAEBMB4wHAYIKwYBBQUHAgEWEHd3dy5zZWN1cm9zeXMuY2g
wDgYDVR0PAQH/BAQDAgFGMA0GCSqGSIb3DQEBCwUAA4IBAQAkMhsXi+A6UOamNzVbemdMAY1eOdZzEScd6iLZQI0
OvOh8IKNYAODIzm7HKNuaqlL/9G9BQPu2MZIzf8MffxudNJzYB6uBFNzxT81kHBBhoDYxZr2Lj0r30jBPLF4Ccnp
NB/S2FinqhWRvG5Yub9JqMPKr/V+f+UIzGTio96gygYWGQQSLhwcn2llDnVspRnB9eTvdckdHL6CXbnfPeYxS0JF
jPEF4FV9s0KZz62TP0XKq1hgKPAJtKvAUrtcRRBLJm2xTnsfelV2LoY5yGRJk5gHr0YIDJuoZIDyhE1LKs2fnl7t
cwEkt6gABizTPevs29hiC8PlX3q4UEPXfyoxz",

"MIIDqzCCApOgAwIBAgIRAJ2TU2SzRdjLZtI48GsWaNEwDQYJKoZIhvcNAQELBQAwXzELMAkGA1UEBhMCQ0gxCzA
JBgNVBAgTAlpIMQ8wDQYDVQQHEwZadXJpY2gxEjAQBgNVBAoTCVNlY3Vyb3N5czEeMBwGA1UEAxQVU0VDVVJPU1l
TX1JPT1RLRVlfREVWMCAXDTIwMDUyODExNTc1OVoYDzIwNTAwNTIxMTE1NzU5WjBfMQswCQYDVQQGEwJDSDELMAk
GA1UECBMCWkgxDzANBgNVBAcTBlp1cmljaDESMBAGA1UEChMJU2VjdXJvc3lzMR4wHAYDVQQDFBVTRUNVUk9TWVN
fUk9PVEtFWV9ERVYwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCmLFqDEqpNXtPjFJtRIwscC0uwUZO
1sdWNZp61WI1hiuVNpT2HNSOVg1AhHWbxbp5UaDvExRbC3jYboi1ybVWHRatfyT6cFuOP8hkkqMnhr62C5btLDqu
9/KOfM6p7RE7BijVpJgTJV+ZSuroB0hLz5a7CMGEJq/UYFbqesUQX7I73w732fZIxhtPSciNgmhIrAzTiuyz+wkM
cequatZxPi5CaoGeFgZxDxXxp0XT7bSatWuuyPtz9oyVchvqZGiC8wS62xl1ehrJTLbq9XHA2LOK35ax68hzNzow
PN96ytpyq91G8zSA8l8MUps2qLkKmZW6sCCvKThTe49cfd331AgMBAAGjYDBeMBIGA1UdEwEB/wQIMAYBAf8CAQE
wDgYDVR0PAQH/BAQDAgFGMDgGA1UdIAQxMC8wLQYLKwYBBAGC3HwEAQEwHjAcBggrBgEFBQcCARYQd3d3LnNlY3V
yb3N5cy5jaDANBgkqhkiG9w0BAQsFAAOCAQEANfnMu0WqGsh6NmgXBJamQB243UcBXxJoJzkdEB9IcwyjEb9c1Kw
A0TXDuwHpyhlPTFKsPHEhg+dRNLsO0NZIQ1ITQE3WI5P1zwEYRm3pSLHDKBFIielk5GC3zYMQ3GoSU6VRdfDob65
```

```
Jol3k3Z/WV/KV7YM+GQJPHPL/+4xYK166L0SKGg6wJwipyo77fErxvbdOuJwDsjWqDIyrd7VLz6LhjRoxN2mMWcI
vB2J0cGAxwc3n48a+9NGGIOFnCfTl7ABniQSGu7iYU50bisVrvypLdd2q+q64nPNZiMkE/ruydw7tiW74QqcYcnM
FjEKVM7OQBSz3bt7c7Zu/tIwV4g=="
  ]
}
```

8.4.2 POST/v1/key

Upon the creation of a key, all relevant key attestation data is automatically returned in the response and doesn't need to be requested separately.

```
curl -X 'POST' \
  'https://tsb-demo.cloudshsm.com/v1/key' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "label": "test-key",
  "algorithm": "RSA",
  "keySize": 2048,
  "attributes": {
    "decrypt": false,
    "sign": true,
    "unwrap": false,
    "destroyable": true
  }
}'
```

Example of a successful operation:

```
{
  "xml": "<private_key creation=\"generated\">\n\t<label>test-
key</label>\n\t<create_time>2022-10-17T08:43:26Z</create_time>\n\t<attest_time>2022-10-
17T08:43:26Z</attest_time>\n\t<algorithm>RSA</algorithm>\n\t<algorithm_oid>1.2.840.11354
9.1.1.1</algorithm_oid>\n\t<key_size>2048</key_size>\n\t<public_key>MIIBIjANBgkqhkiG9w0B
AQEFAAOCAQ8AMIIBCgKCAQEA0MX3kihx6DP9H5KiF7pdhEs9FvKrHqCy6JWjb4GP0zHqrq2G5Ucc4aWh8a+QE5zu
kYghqGyi0BJ+nAVD2Sp+QVaOqMZBASj/lqgJPGBaISepKuzFExVHDcTRi5HBTjZpmU8fyEC74OZ8ehRoh4U5nrn0
jZ4GqzWxS07msyUccHRhFfvcHtJjg9ssfNX08roAK+0hBozPjLU9Vhdc49FA17O6Gq+5s4T2lGNfQXJnsrZ9VlYU
MqxKNabLG9tocy5IAV6YvV9axSY6S1J6df7Olj/wXDfWSP5FvX+bM1QydGIL9ImELYsHlZfKwL8L1uH95CkmfecW
+SPRp1ihtlrbWQIDAQAB</public_key>\n\t<attributes>\n\t\t<decrypt>false</decrypt>\n\t\t<si
gn>true</sign>\n\t\t<unwrap>false</unwrap>\n\t\t<derive>false</derive>\n\t\t<sensitive>t
rue</sensitive>\n\t\t<always_sensitive>true</always_sensitive>\n\t\t<extractable>false</
extractable>\n\t\t<never_extractable>true</never_extractable>\n\t\t<modifiable>false</mo
difiable>\n\t\t<copyable>false</copyable>\n\t\t<destroyable>true</destroyable>\n\t\t<uni
que>true</unique>\n\t</attributes>\n</private_key>\n\n",
  "json": {
    "label": "test-key",
    "id": null,
    "algorithm": "RSA",
    "algorithmOid": "1.2.840.113549.1.1.1",
    "curveOid": null,
    "keySize": 2048,
    "createTime": "2022-10-17T08:43:26Z",
    "attestTime": "2022-10-17T08:43:26Z",
    "publicKey":
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0MX3kihx6DP9H5KiF7pdhEs9FvKrHqCy6JWjb4GP0zH
qrq2G5Ucc4aWh8a+QE5zukYghqGyi0BJ+nAVD2Sp+QVaOqMZBASj/lqgJPGBaISepKuzFExVHDcTRi5HBTjZpmU8
fyEC74OZ8ehRoh4U5nrn0jZ4GqzWxS07msyUccHRhFfvcHtJjg9ssfNX08roAK+0hBozPjLU9Vhdc49FA17O6Gq+
5s4T2lGNfQXJnsrZ9VlYUMqxKNabLG9tocy5IAV6YvV9axSY6S1J6df7Olj/wXDfWSP5FvX+bM1QydGIL9ImELYs
HlZfKwL8L1uH95CkmfecW+SPRp1ihtlrbWQIDAQAB",
    "addressTruncated": null,
    "attributes": {
      "decrypt": false,
      "sign": true,
      "ekaSign": null,
      "unwrap": false,
      "derive": false,
      "sensitive": true,
```

```
      "alwaysSensitive": true,
      "extractable": false,
      "neverExtractable": true,
      "modifiable": false,
      "copyable": false,
      "destroyable": true
    },
    "policy": null
  },
  "xmlSignature":
"pgw9xNA3kQ2kAR+EY3EGktEXMhrQ4DrweORYmnK4YCn1VzlRsytLkjy23QLon2gzMNTzPgnp2ab2kWHV0Z0MZ4r
yf9F/MI7PRNU+wPZEB8N1veaWSUSg/xxGmmiYujsJ1Ij6oLMgMzJMZrrhQqcMfuIScUeZr6wOLP6XeTrQH6SnRxN
RwXqw2nLt4mGgVRfUrPfpiGY5n5esJkoPEBubo9mc20QmYbusS0HrWWPdNXid4E8fvz5UOAQ4bS09JGr03DU2lyG
LOLEbhdcMYjGcJdxUv09JS+bagoXc0Dd7fUTVBDvM3q4C9+RWt8St1H2eRdrlUy7HnRxy+e6VYldfSA==",
  "attestationKeyName": "attestation-key"
}
```


8.4.3 GET/v1/key/{keyName}/attributes

All relevant key attestation data of a given key can be requested as well at any point of time after the key was generated.

```
curl -X 'GET' \
  'https://tsb-demo.cloudshsm.com/v1/key/test-key/attributes' \
  -H 'accept: application/json'
```

Example of a successful operation:


```
{
  "xml": "<private_key creation=\"generated\">\n\t<label>test-
key</label>\n\t<create_time>2022-10-17T08:43:26Z</create_time>\n\t<attest_time>2022-10-
17T08:50:03Z</attest_time>\n\t<algorithm>RSA</algorithm>\n\t<algorithm_oid>1.2.840.11354
9.1.1.1</algorithm_oid>\n\t<key_size>2048</key_size>\n\t<public_key>MIIBIjANBgkqhkiG9w0B
AQEFAAOCAQ8AMIIBCgKCAQEA0MX3kihx6DP9H5KiF7pdhEs9FvKrHqCy6JWjb4GP0zHqrq2G5Ucc4aWh8a+QE5zu
kYghqGyi0BJ+nAVD2Sp+QVaOqMZBASj/lqgJPGBaISepKuzFExVHDcTRi5HBTjZpmU8fyEC74OZ8ehRoh4U5nrn0
jZ4GqzWxS07msyUccHRhFfvcHtJjg9ssfNX08roAK+0hBozPjLU9Vhdc49FA17O6Gq+5s4T2lGNfQXJnsrZ9VlYU
MqxKNabLG9tocy5IAV6YvV9axSY6S1J6df7Olj/wXDfWSP5FvX+bM1QydGIL9ImELYsHlZfKwL8L1uH95CkmfecW
+SPRp1ihtlrbWQIDAQAB</public_key>\n\t<attributes>\n\t\t<decrypt>false</decrypt>\n\t\t<si
gn>true</sign>\n\t\t<unwrap>false</unwrap>\n\t\t<derive>false</derive>\n\t\t<sensitive>t
rue</sensitive>\n\t\t<always_sensitive>true</always_sensitive>\n\t\t<extractable>false</
extractable>\n\t\t<never_extractable>true</never_extractable>\n\t\t<modifiable>false</mo
difiable>\n\t\t<copyable>false</copyable>\n\t\t<destroyable>true</destroyable>\n\t\t<uni
que>true</unique>\n\t</attributes>\n</private_key>\n\n",
  "json": {
    "label": "test-key",
    "id": null,
    "algorithm": "RSA",
    "algorithmOid": "1.2.840.113549.1.1.1",
    "curveOid": null,
    "keySize": 2048,
    "createTime": "2022-10-17T08:43:26Z",
    "attestTime": "2022-10-17T08:50:03Z",
    "publicKey":
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0MX3kihx6DP9H5KiF7pdhEs9FvKrHqCy6JWjb4GP0zH
qrq2G5Ucc4aWh8a+QE5zukYghqGyi0BJ+nAVD2Sp+QVaOqMZBASj/lqgJPGBaISepKuzFExVHDcTRi5HBTjZpmU8
fyEC74OZ8ehRoh4U5nrn0jZ4GqzWxS07msyUccHRhFfvcHtJjg9ssfNX08roAK+0hBozPjLU9Vhdc49FA17O6Gq+
5s4T2lGNfQXJnsrZ9VlYUMqxKNabLG9tocy5IAV6YvV9axSY6S1J6df7Olj/wXDfWSP5FvX+bM1QydGIL9ImELYs
HlZfKwL8L1uH95CkmfecW+SPRp1ihtlrbWQIDAQAB",
    "addressTruncated": null,
    "attributes": {
      "decrypt": false,
      "sign": true,
      "ekaSign": null,
      "unwrap": false,
      "derive": false,
      "sensitive": true,
      "alwaysSensitive": true,
```

```
      "extractable": false,
      "neverExtractable": true,
      "modifiable": false,
      "copyable": false,
      "destroyable": true
    },
    "policy": null
  },
  "xmlSignature":
"Wz8Gby94kIy8IF8nUNn+kGfzCh9P859uunNiVJRDUaTSXUlF/g/gy42iaPxy91FnGnmFIjCvRtfOV3gmnotv+Ds
hk46Qdyl4DUqKUELX7T1QDpKsU64J+My1XBnYMkEo21cIPktfQJ1iyEkVzZW3WrvUG3mfOR1v5jyoD4TCFgzejgo
vPtCL9vOCRaT7YQoo4BN8wIcmU/Cc41Yy68hgigW2/p0jes5pgMZXGWaZ1pwfDYyBASAN+YRoMkQDYQS7EvxbGcp
TtlC59OOhihH0wbsv2zJSSqAhRKj3OcKm24NaPmCI1Lxw/7yzhWZzHfNN3rcZCMGozvdAo0oDgZGRwQ==",
  "attestationKeyName": "attestation-key"
}
```