HKG18-402: Secure Key Services in OP-TEE

Etienne Carrière, STMicroelectronics
Agenda

● Overview of Secure Key Services
● What is OP-TEE?
● Which Client Interface?
● Implementation: the SKS
● What’s next?
HSM, SE, TPM, TEE: Secure Services

- **Hardware Security Module (HSM)**

- **Secure Elements as Smartcard, SIM cards.**
  
  [https://www.globalplatform.org/mediaguideSE.asp](https://www.globalplatform.org/mediaguideSE.asp)

- **Trusted Platform Modules (TPM devices)**
  
  [https://trustedcomputinggroup.org/work-groups/trusted-platform-module/](https://trustedcomputinggroup.org/work-groups/trusted-platform-module/)

- **Trusted Execution Environment (TEE)**
  
  Several standards including the GPD TEE
  
  [https://www.globalplatform.org/specificationsdevice.asp](https://www.globalplatform.org/specificationsdevice.asp)

Pictures: source wikipedia (public domain) and ARM ©
HSM, SE, TPM, TEE: Secure Keys

HSMs, SEs, TPMs provide secure key management services:

- Key materials and cryptographic operations are very hard to tamper with.
- Client can import, generate, derive keys and cipher, sign, authenticate data.
- Secure keys have usage constraints.
- Use of secure keys may require user authentication.

How can the open source help in secure key management services?

TEEs as OP-TEE are suitable to propose such HSM services.
Sadly there is no uniform interface on which OP-TEE could build such a service.
Agenda

- Overview of Secure Key Services
- What is OP-TEE?
- Which Client Interface?
- Implementation: the SKS
- What’s next?
What is OP-TEE?

- Open source Trusted Execution Environment for Armv7/Armv8-A platforms.
- OP-TEE relies on the GPD TEE Client API specifications.
- OP-TEE relies on the GPD TEE Internal Core API specifications.

https://www.globalplatform.org/specificationsdevice.asp
What is OP-TEE?

OP-TEE relies on the GPD TEE Client and Internal Core API specifications

Step 1: Client opens a session toward a trusted application (TA).

➤ Trusted application identifies client and returns a session handle.

Step 2: Client invokes TA commands each with up to 4 parameters.

➤ Trusted application checks the 32bit command ID and its parameters.
➤ Trusted application executes the command.
➤ Trusted application returns a status, eventually output data.

Step 3: Client closes the session.
What is OP-TEE?

GPD TEE Internal Core API functions for secure storage and cryptography:

- **Secure Storage relates functions**
  - TEE_CreatePersistentObject(), TEE_OpenPersistentObject(),
  - TEE_CloseAndDeletePersistentObject1(), TEE_ReadObjectData(),
  - TEE_WriteObjectData(), TEE_TruncateObjectData(), TEE_SeekObjectData().

- **Cryptographic operations functions**
  - TEE_DigestInit/Update/DoFinal(), TEE_CipherInit/Update/DoFinal(),
  - TEE_MACInit/Update/ComputeFinal/CompareFinal(), TEE_DeriveKey(),
  - TEE_AEInit/AEUpdateAAD/AEUpdateAAD/AEEncryptFinal/AEDecryptFinal(),
  - TEE_AsymmetricEncrypt/Decrypt/SignDigest/VerifyDigest(), TEE_GenerateKey().
Agenda

● Overview of Secure Key Services
● What is OP-TEE?
● Which Client Interface?
● Implementation: the SKS
● What’s next?
Secure Key Services: Which Client Interface?

Userland Applications

- Client Application
- Client API library
- Secure key services
- Linux crypto layer
- Secure key crypto provider
- OP-TEE Client API Library
- OP-TEE Client API
- OP-TEE Linux driver
- OP-TEE Core Firmware
- Secure Key Services Trusted Application
- OP-TEE Internal Core API Library
- Secure world
- Normal world

Bootloader

- Some Bootloader Executing in Non Secure State
- Secure Keys API
- OP-TEE Client API
- OP-TEE Supplicant
- Normal world

Kernel crypto services

... also for Trusted Applications?
Secure Key Services: Which Client Interface?

Linux kernel services:

- Requires integration of the kernel client API in OP-TEE Linux driver.

Bootloader clients

- An OP-TEE portable client library to leverage TEE from bootloaders?
- OP-TEE secure storage currently relies on physical media access through REE. Secure key service at boot implies TEE supplicant services in the bootloader.
Secure Key Services: Which Client Interface?

Userland clients: which API?

- Mainly proprietary libraries and interfaces in vendor solutions.
- TPM Interface ([https://trustedcomputinggroup.org/tpm-library-specification](https://trustedcomputinggroup.org/tpm-library-specification))
  - TPM already comes with an integration framework (TSS).
  - TPM lacks secure time to bound object time validity.
  - Very rich featured API but requires some of the Android support.
  - Quite rich and extendable interface.
- Others libraries or APIs defined in the open source community?
Secure Key Services: Which Client Interface?

Lot of convergence of Android Keystore and PKCS #11 APIs:
- Crypto algorithms, operations atomicity, objects generic attributes.

PKCS #11:
- Referenced in many frameworks (i.e. simalliance, amazon-freertos, linuxonibm).
- User authentication is restricted to a Security Officer and a single user.
- Flexible for extensions of object attributes and crypto schemes.

Android Keystore (far not exhaustive):
- More key attributes and rich binding with client application identity.
- Attestation of keys and device information using certificates.
Agenda

- Overview of Secure Key Services
- What is OP-TEE?
- Which Client Interface?
- Implementation: the SKS
- What’s next?
OP-TEE SKS Proposal

PKCS #11 API userland library
➔ github.com/OP-TEE/optee_client/libsks/

Trusted Application: the SKS TA
➔ github.com/OP-TEE/optee_os/ta_services/secure_key_services/

OP-TEE regression test environment
➔ github.com/OP-TEE/optee_test/host/xtest/regression_xxxx.c
OP-TEE SKS Proposal

First start by a workbase delivery:

- Very reduced cryptographic support (AES flavors, maybe a bit of RSA or ECC).
- Reduced set of PKCS #11 functions.
- Integration with Linux userland applications only.

Then will come more cryptographic support.
Then will come kernel and bootloader interfaces.
Then will come extended object attributes?

Contributions will be welcome!
OP-TEE SKS Proposal

PKCS #11 Cryptographic Token - [https://www.oasis-open.org/committees/pkcs11/](https://www.oasis-open.org/committees/pkcs11/)

Latest as of today (March 2018) is the Specifications Version 2.40 Plus Errata 01:

- **The Interface Base Specification** defines the functions and most ABI.
- **The Current Mechanisms Specification** lists mechanisms and their parameters.
- **The Interface Historical Mechanisms** lists historical mechanisms (i.e. DES).
- **Three C/C++ code header files:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Last Modified</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>pkcs11.h</td>
<td>13-May-2016 16:00</td>
<td>8k</td>
</tr>
<tr>
<td>pkcs11f.h</td>
<td>13-May-2016 16:00</td>
<td>27k</td>
</tr>
<tr>
<td>pkcs11t.h</td>
<td>13-May-2016 16:00</td>
<td>70k</td>
</tr>
</tbody>
</table>
PKCS #11 defines API functions and their arguments.
The SKS TA API defines one command per PKCS #11 function.
TA command parameters reflect the PKCS #11 function arguments.

- C_InitToken() → SKS_CMD_CK_INIT_TOKEN
- C_CreateObject() → SKS_CMD_IMPORT_OBJECT
- C_EncryptInit() → SKS_CMD_ENCRYPT_INIT
- C_EncryptUpdate() → SKS_CMD_ENCRYPT_UPDATE
- C_EncryptFinal() → SKS_CMD_ENCRYPT_FINAL
- C_CloseSession() → SKS_CMD_CK_CLOSE_SESSION
Objects: An object is a collection of attributes

- Class and type in class, i.e. a symmetric key for an AES processing.
- Secret object secret value(s), i.e. an AES key value.
- Identification means: label, ID, and very few others.
- Storage attributes: persistent, non modifiable, etc...
- Use constraints: allowed operations, time validity, user authentication, etc...

**S.SK S TA API**

**Objects:** An object is a collection of **attributes**.

An attribute is a triplet **attribute-ID/value-byte-size/value-data**.

![Diagram of object representation in PKCS#11 ABI](image1)

Representation of an object in the PKCS#11 ABI

![Diagram of object representation in SKS TA ABI](image2)

Representation of an object in the SKS TA ABI
Mechanisms are cryptographic operation schemes defined by:

- An identification number;
- Formatted parameters required to initialize a crypto operation;
- Ability to execute processing modes or functions (i.e. encrypt, sign, derive).

Examples of PKCS #11 mechanisms:

- AES MAC: CKM_AES_MAC, no parameter, supports sign and verify.
- AES CBC: CKM_AES_CBC, requires an IV, supports encryption and decryption.
- AES GCM: CKM_AES_GCM, requires an IV, an AAD and a tag size, can be used to encrypt and decrypt Authenticated Encryption (AE) messages.
SKS TA API

Format of mechanism parameters in the PKCS #11 and SKS TA ABIs

AES MAC has no parameter

<table>
<thead>
<tr>
<th>CK_ULONG mechanism;</th>
</tr>
</thead>
<tbody>
<tr>
<td>CK_VOID *params;</td>
</tr>
<tr>
<td>CK_ULONG param_size;</td>
</tr>
</tbody>
</table>

= NULL
= 0

32bit ID 32bit size = 0

head

AES CBC needs a 16-byte IV

<table>
<thead>
<tr>
<th>CK_ULONG mechanism;</th>
</tr>
</thead>
<tbody>
<tr>
<td>CK_VOID *params;</td>
</tr>
<tr>
<td>CK_ULONG param_size;</td>
</tr>
</tbody>
</table>

= 16

32bit ID 32bit size = 16 16byte initial vector

head parameters data

AES GCM requires a structured parameters set

<table>
<thead>
<tr>
<th>CK_ULONG mechanism;</th>
</tr>
</thead>
<tbody>
<tr>
<td>CK_VOID *params;</td>
</tr>
<tr>
<td>CK_ULONG param_size;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CK_BYTE *iv;</th>
</tr>
</thead>
<tbody>
<tr>
<td>CK_ULONG iv_byte_size;</td>
</tr>
<tr>
<td>CK_BYTE *aad;</td>
</tr>
<tr>
<td>CK_ULONG aad_byte_size;</td>
</tr>
<tr>
<td>CK_ULONG tag_bit_size;</td>
</tr>
</tbody>
</table>

uint8_t iv[iv_byte_size]
uint8_t aad[aad_byte_size]

32b ID 32b size 32b IV size IV data 32b AAD size AAD data 32b tag size

head parameters data
SKS TA: Processing

Processing functions follow the same sequence in the SKS trusted application:

- Get/check parameters.
- Check function against session state.
- Prepare created key (if any) attribute list.
- Check created key (if any) against session state.
- Check created key (if any) against function.
- Check used key (if any) against session state.
- Check used key (if any) against function.
- Process requested crypto operation ➞ wrap to GPD TEE crypto API.
- Register created key (if any).
- Return a status and an object handle or processed data.

C_GenerateKey()
C_DeriveKey()
C_EncryptInit()
C_GenerateKey()
C_VerifyUpdate()
Agenda

- Overview of Secure Key Services
- What is OP-TEE?
- Which Client Interface?
- Implementation: the SKS

- What’s next?
Current Status

As of March 2018:

- Proposed an API for the TA that reflects the PKCS #11 API.
- Supports token info retrieve and sessions functions.
- Supports persistent storage of keys and token state.
- Import and generation of generic secrets and AES keys.
- AES in modes ECB, CBC, CTS, CTR, GCM and CCM.
- SHAxxx HMACs, AES CMAC, AES CBC MAC.
- Several token management and set/get attributes functions are not supported.
- Test environment still weak.
What’s Next - Short Term

- Enhance the tests and constraints on client parameters. Existing PKCS #11 test frameworks?
- Consider delivery in OP-TEE 3.1.0 if mature enough.
- Contributions will be welcome to enhance the set of crypto algorithms and mature the implementation.
What’s Next - Long Term

- Issue #1: certificate support
  - Current OP-TEE does not provide any certificate support.
  - Secure parsing of X.509 certificates is known to be touchy.

- How to provision the SKS key database with platform secrets (i.e. OTP fuses)?

- Interface keys and operations deported in a more secure backend HSM/SE?

- Consider convergences with an Android keystore solution.

- Looking forward PKCS #11 Version 3.0 ([cryptsoft.com](http://cryptsoft.com)).
What’s Next

- Integrate in a filesystem encryption setup
  - May requires SKS TA services at boot stage (ongoing work by Igor Opaniuk for a portable OP-TEE client library).
  - Requires the OP-TEE kernel client API in the Linux optee driver.
  - SKS TA can be a transformation providers in the Linux Crypto API.
What’s Next - Short Term

Provisioning and token ownership
- Clarify the provisioning sequences.
- How should we handle several PKCS #11 tokens?
- Can we create on request provisioned tokens with delegated ownership?

Release the SKS TA during long lasting operations
- An OP-TEE TA is not reentrant.
- Could the SKS TA delegate a crypto processing to a TA instance?
Thank You
etienne.carriere@linaro.org [github: @etienne-lms]

#HKG18

HKG18 keynotes and videos on: connect.linaro.org
For further information: www.linaro.org
Some extra slides...
PKCS #11 Attributes (1/3)

- Attribute CKA_CLASS
  CKO_DATA, CKO_CERTIFICATE, CKO_PUBLIC_KEY, CKO_PRIVATE_KEY,
  CKO_SECRET_KEY, CKO_HW_FEATURE, CKO_DOMAIN_PARAMETERS,
  CKO_MECHANISM, CKO_OTP_KEY, CKO_VENDOR_DEFINED.

- Attribute CKA_KEY_TYPE
  CKK_RSA, CKK_DSA, CKK_DH, CKK_EC, CKK_AES, CKK_SHA256_HMAC,
  CKK_HOTP, CKK_DES3, CKK_GENERIC_SECRET, and more.

- Attribute CKA_CERTIFICATE_TYPE
  CKC_X_509, CKC_X_509_ATTR_CERT, CKC_WTLS, CKC_VENDOR_DEFINED
PKCS #11 Attributes (2/3)

- Boolean Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_TOKEN</td>
<td>CKA_PRIVATE</td>
<td>CKA_ALWAYS_AUTHENTICATE</td>
</tr>
<tr>
<td>CKA_ENCRYPT</td>
<td>CKA_DECRYPT</td>
<td>CKA_DERIVE</td>
</tr>
<tr>
<td>CKA_WRAP</td>
<td>CKA_UNWRAP</td>
<td></td>
</tr>
<tr>
<td>CKA_SIGN</td>
<td>CKA_SIGN_RECOVER</td>
<td></td>
</tr>
<tr>
<td>CKA_VERIFY</td>
<td>CKA_VERIFY_RECOVER</td>
<td></td>
</tr>
<tr>
<td>CKA_EXTRACTABLE</td>
<td>CKA_NEVER_EXTRACTABLE</td>
<td></td>
</tr>
<tr>
<td>CKA_SENSITIVE</td>
<td>CKA_ALWAYS_SENSITIVE</td>
<td></td>
</tr>
<tr>
<td>CKA_MODIFIABLE</td>
<td>CKA_COPYABLE</td>
<td>CKA_DESTROYABLE</td>
</tr>
<tr>
<td>CKA_LOCAL</td>
<td>CKA_TRUSTED</td>
<td>CKA_WRAP_WITH_TRUSTED</td>
</tr>
</tbody>
</table>
PKCS #11 Attributes (3/3)

- Other Attributes
  - CKA_VALUE, CKA_VALUE_LEN,
  - CKA_LABEL, CKA_OBJECT_ID, CKA_APPLICATION, CKA_ID,
  - CKA_START_DATE, CKA_END_DATE,
  - CKA_WRAP_TEMPLATE, CKA_UNWRAP_TEMPLATE, CKA_DERIVE_TEMPLATE,
  - CKA_MODULUS, CKA_PRIVATE_EXPONENT, CKA_PRIME, CKA_EC_PARAMS, etc...,
  - CKA_CERTIFICATE_CATEGORY, CKA_ISSUER, CKA_SERIAL_NUMBER, etc...,
  - CKA_OTP_FORMAT, CKA_OTP_LENGTHCKA_OTP_TIME_INTERVAL,
  - CKA_KEY_GEN_MECHANISM, CKA_LOCAL,
  - CKA_MECHANISM_TYPE, CKA_ALLOWED_MECHANISMS,
  - CKA_VENDOR_DEFINED
Android Keystore: Object Attributes

- Same common attributes as PKCS #11 for algo/function constraints on keys.
- PIN/password assignment per key.
- Key wrapped inside the secure device.
- Secret can be bound to boot stages.
- Specific encrypt/sign and decrypt/verify expiration dates.
- Bandwidth restrictions, access count restrictions.

https://source.android.com/security/keystore/tags
Android Keystore: Object Attributes

Tag::PURPOSE  Tag::ALGORITHM  Tag::KEY_SIZE
Tag::UNIQUE_ID  Tag::APPLICATION_DATA  Tag::APPLICATION_ID
Tag::BLOB_USAGE_REQUIREMENTS  Tag::BOOTLOADER_ONLY
Tag::MAXUSES_PER_BOOT  Tag::MINSECONDS_BETWEEN_OPS
Tag::USER_SECURE_ID  Tag::USER_AUTH_TYPE  Tag::NO_AUTH_REQUIRED
Tag::ORIGINATION_EXPIRE_DATETIME  Tag::USAGE_EXPIRE_DATETIME
...

https://source.android.com/security/keystore/tags
Android Keystore: API Functions

- Import/export raw symmetric keys and formatted asymmetric keys.
- Generate and derive keys and usual ciphering and/or authentication algo.
- Attest a key: export a certificate for a given key.
- Attest a device hardware information in a certificate.

`addRngEntropy()`, `getHardwareFeatures()`, `generateKey()`, `importKey()`, `getKeyCharacteristics()`, `exportKey()`, `deleteKey()`, `deleteAllKeys()`, `destroyAttestationIds()`, `begin()`, `update()`, `finish()`, `abort()`.

[https://source.android.com/security/keystore/implementer-ref](https://source.android.com/security/keystore/implementer-ref)
End of extras
etienne.carriere@linaro.org

#HKG18
HKG18 keynotes and videos on: connect.linaro.org
For further information: www.linaro.org