Unified TEE Management API

The following document outlines how the current solution devised by the IETF TEEP effort would be affected by changing the concept like proposed at: <https://github.com/ietf-teep/architecture/issues/52>

The current solution is an API expressed in JSON. A similar API expressed in CBOR has also been mentioned as desirable.

This proposal defines a unified binary level API, independent of JSON and CBOR. The proposal builds on predecessors like TPM, PKCS #11 and ISO7816. As an example, smart cards are often [locally] personalized through a card-specific shared secret. This also serves as “attestation” since the card is verifiably authentic.

With respect to IPR, the only thing that might be “novel” is the *combination of*

* Shared secret creation through ECDH
* Device attestation

which was filed 2012 as a *defensive publication*: <https://priorart.ip.com/IPCOM/000215433>

An obvious advantage of a low-level binary API is that it enables basic *compile-time type checking*. A JSON-based API requires run-time type checking.

When the low-level binary API is called through a JSON based protocol, run-time checking of JSON structures and conversions are performed by the REE.

# Session Creation

This is currently TBD but the basics include:

* Session key creation using ECDH with at least the TEE-side using an ephemeral key
* TEE attestation that would also include the ephemeral key
* Creation of a non-secret session ID used in all API calls

# *Continued on the next page…*InstallTA – Current Solution

The current solution requires a rather quirky two level JSON scheme for the “call” part of the API.

**Outer message:**

**{**

**"InstallTARequest": {**

**"payload": "<InstallTATBSRequest JSON above>",**

**"protected": "<integrity-protected header contents>",**

**"header": "<non-integrity-protected header contents>",**

**"signature": "<signature contents signed by TAM private key>"**

**}**

**}**

**Inner message:**

**{**

**"InstallTATBSRequest": {**

**"ver": "1.0",**

**"rid": "<unique request ID>",**

**"tid": "<transaction ID>",**

**"tee": "<TEE routing name from the DSI for the SD's target>",**

**"nextdsi": true | false,**

**"dsihash": "<hash of DSI returned in the prior query>",**

**"content": ENCRYPTED {**

**"tamid": "<TAM ID previously assigned to the SD>",**

**"spid": "<SPID value>",**

**"sdname": "<SD name for the domain to install the TA>",**

**"spcert": "<BASE64 encoded SP certificate >", // optional**

**"taid": "<TA identifier>"**

**},**

**"encrypted\_ta": {**

**"key": "<JWE enveloped data of a 256-bit symmetric key by**

**the recipient's TEEspaik public key>",**

**"iv": "<hex of 16 random bytes>",**

**"alg": "<encryption algoritm. AESCBC by default.",**

**"ciphertadata": "<BASE64 encoded encrypted TA binary data>",**

**"cipherpdata": "<BASE64 encoded encrypted TA personalization data>"**

**}**

**}**

**}**

This arrangement comes with the following caveat from the draft:

*The top element "<name>[Signed][Request|Response]" cannot be fully*

*trusted to match the content because it doesn't participate in the*

*signature generation. However, a recipient can always match it with*

*the value associated with the property "payload". It purely serves*

*to provide a quick reference for reading and method invocation.*

# *Continued on the next page…*InstallTA – Proposed Solution

Using the proposed solution the “call” could be something like the following if expressed in JSON:

**{**

**"InstallTARequest": {**

**"sessionid": "<clear text session ID>",**

**"ver": "1.0",**

**"rid": "<unique request ID>",**

**"tid": "<transaction ID>",**

**"tee": "<TEE routing name from the DSI for the SD's target>",**

**"nextdsi": true | false,**

**"dsihash": "<hash of DSI returned in the prior query>",**

**"tamid": "<TAM ID previously assigned to the SD>",**

**"spid": "<SPID value>",**

**"sdname": "<SD name for the domain to install the TA>",**

**"spcert": "<BASE64 encoded SP certificate >", // optional**

**"taid": "<TA identifier>"**

**"ciphertadata": "<BASE64 encoded encrypted TA binary data>",**

**"cipherpdata": "<BASE64 encoded encrypted TA personalization data>",**

**"mac": "<BASE64 HMAC signature derived from the session key>"**

**}**

**}**

The TEE Management API itself could in a Java-like fashion be like:

ReturnValue **InstallTARequest**(String sessionid,

String ver,

String rid,

String tid,

String tee,

boolean nextdsi,

byte[] dsihash,

String tamid,

String spid,

String sdname,

byte[] spcert,

String taid,

byte[] ciphertadata,

byte[] cipherpdata,

byte[] mac)

*Notes*:

Return value: see [Return Values](#_Return_Values)

The API call is *signed* by a HMAC operation over the concatenation of:

* The method name
* The sessionid
* An internal counter which is incremented for each call to check sequence adherence
* All parameters except for mac (which is holding the result)

The key used by the HMAC is derived from the shared session key.

Encrypted parameters like ciphertadata are encrypted by a symmetric key derived from the shared session key.

It is quite possible that a bunch of these parameters like **"rid"** and **"tid"** would rather be associated with the sessionid.

# Session Termination

To “commit” the calls performed during a session, the session must be terminated using a new API method.

The session termination method returns an attestation based on the shared session key telling that the operation succeeded; else it returns an error message.

# Return Values

In order to simplify decoding, return values follow a common scheme based on an object here expressed in Java but would in a real implementation preferably be in CBOR:

**class ReturnValue {**

**boolean success = true;**

**// If success is true, zero or more method specific elements follows**

**// If success is false, a common error object follows**

**}**

Method specific data is always attested by a HMAC signature.

Note that some methods like InstallTA do not seem to need any specific return data.

Author: Anders Rundgren

Version: 0.2