Symmetric key based device attestation

Trusted Firmware M

Tamas Ban
Arm

Agenda

- Attestation service overview
- Token encoding: CBOR and COSE
- Comparison of ECDSA and HMAC auth. tag

What?

Attestation tokens are small reports that are produced by a device upon request. Tokens are composed of key/value pairs called **claims.**

Why?

Device can prove its identity and relying party can assess the device trustworthiness based on the hardware and firmware related claims in the token.

How?

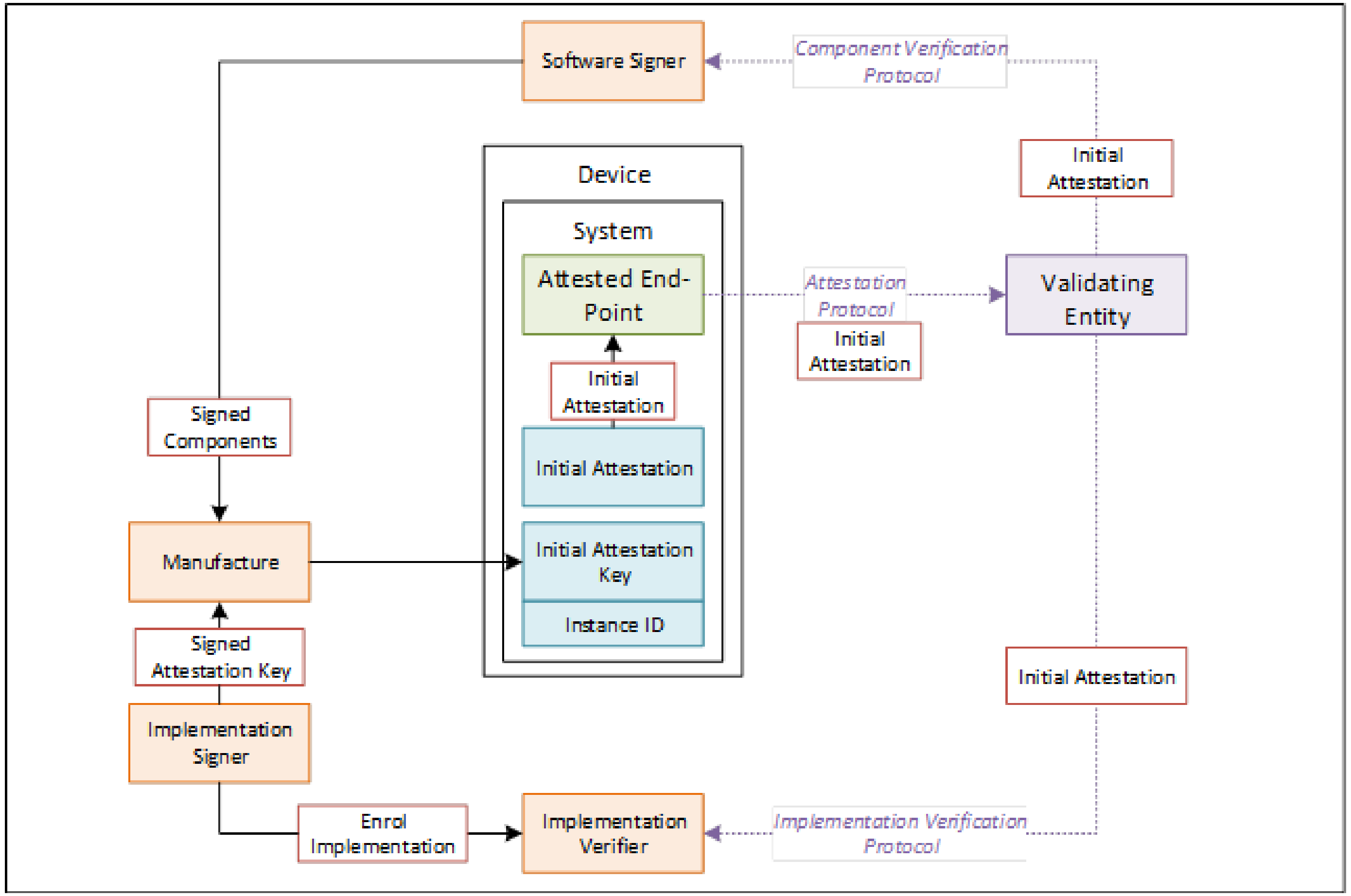
The tokens are *attested* because they are signed by devices using a device-unique cryptographic key. Simple flow:

- Receive an attestation request from the outside world.
- Collect any relevant data, build a report as a set of key/value pairs.
- Format the report in a canonical form and sign it with the device attestation key.
- Send the result back.

arm

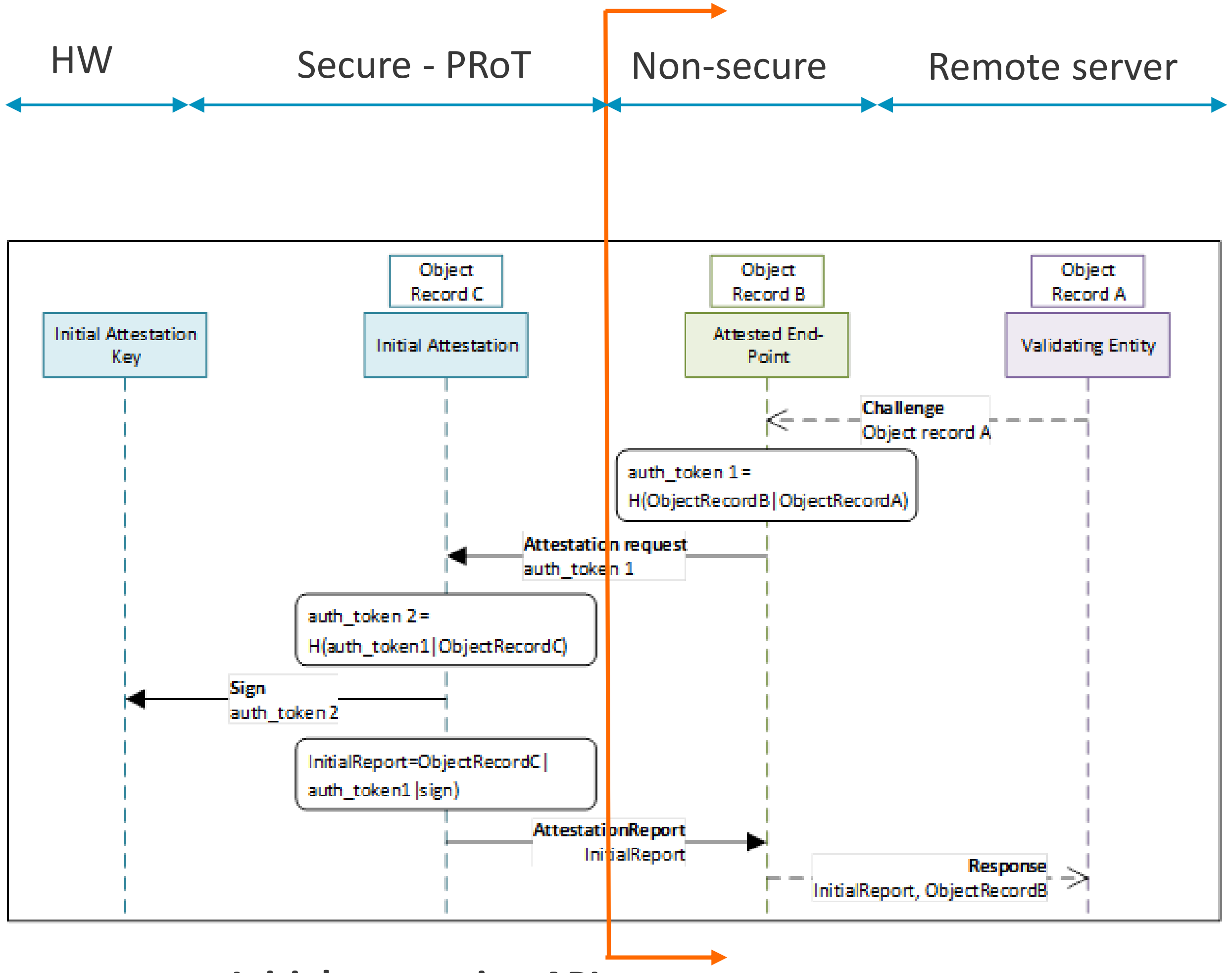
Attestation overview

- Device-unique cryptographic key is securely provisoned during manfacturing
- Verification key and HW ID is extracted and registered to database
- Firmware versions and their measurments value also loaded to the database
- Validation entity checks the token signature and compare claims against database



Attestation flow

- Attestation request received from a remote party
- Challenge can be nonce from server to ensure freshness of the token or locally attested data
- Devic specific data added to the token
- Token authentication tag generated:
  - Asymmetric key: ECDSA P256 over SHA256
  - OR symmetric key: HMAC

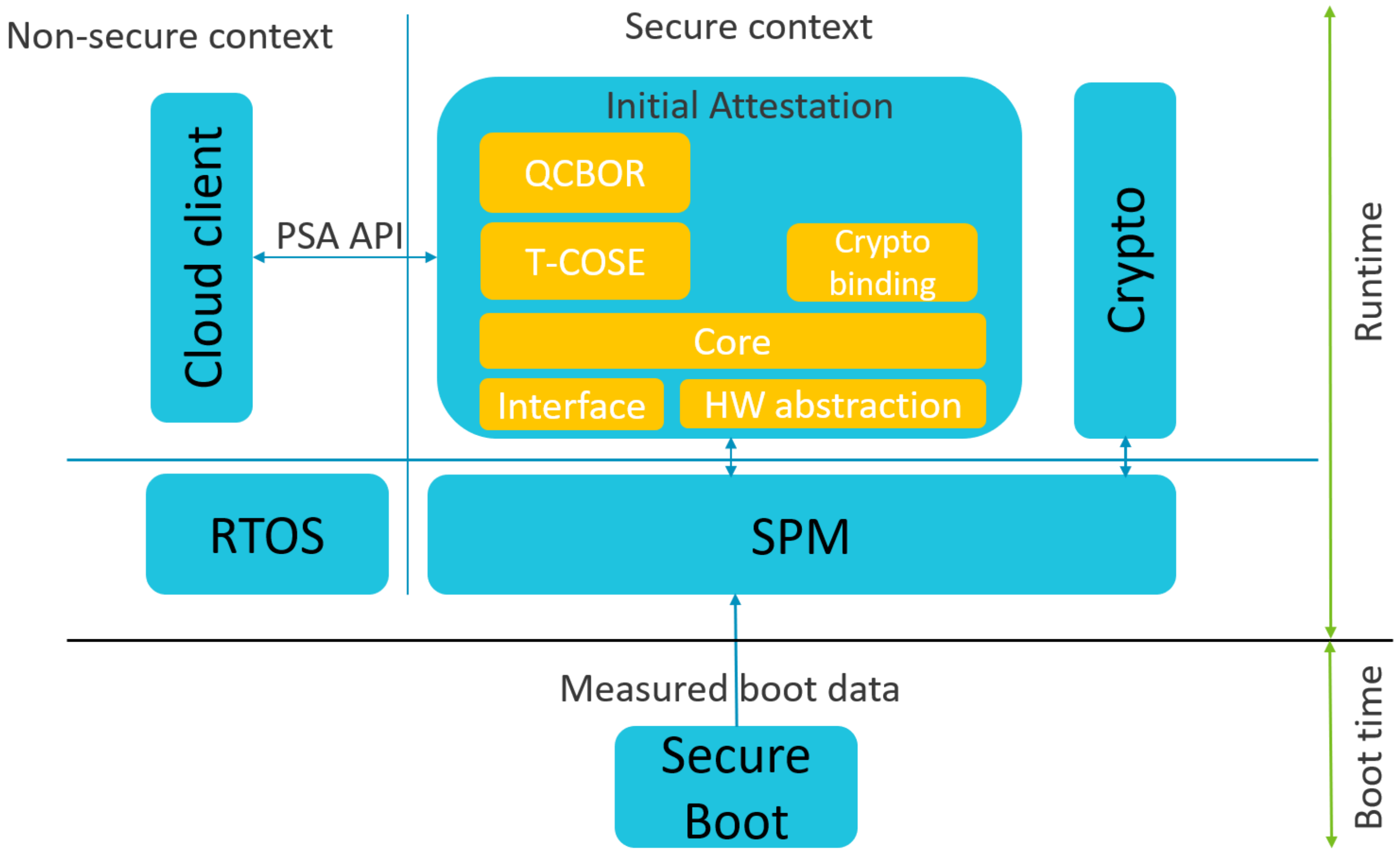


Initial attestation API:

```
psa_initial_attest_get_token(...)
psa_initial_attest_get_token_size(...)
tfm_initial_attest_get_public_key(...)
```

Attestation architecture in TF-M

- Secure bootloader authenticates the firmware images and provide the boot record to runtime firmware to include it to attestation token
- Attestation service collects the data items, encode them to CBOR format and sign the token



arm

CBOR

"Concise Binary Object Representation" (CBOR, http://cbor.io)

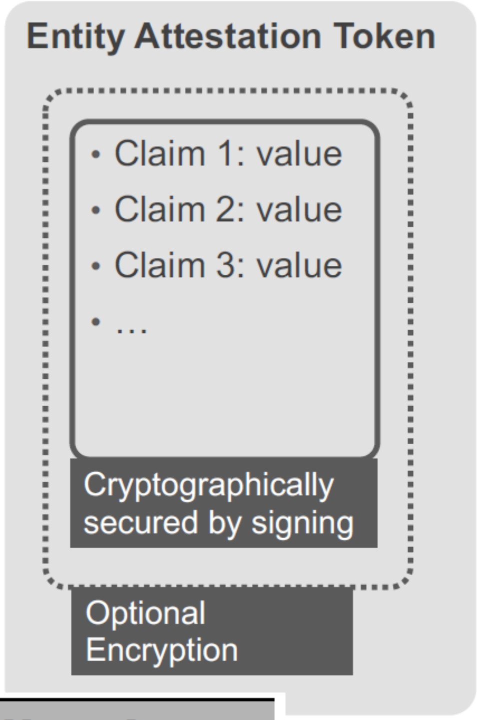
Compact code and data representation for IoT

Standards based (RFC 7049), quite mature

Handles multiple data types, with open source implementations and tools

Data types are simple & powerful – a claim can be a simple integer or have a complex internal structure; allows for optional data

QCBOR library



Four Aspects of Standardization

1. General Structuring and Representation of Claims
   - Labeling of claims
   - Optionality of claims
   - Flexible data representation – integers, strings, binary…

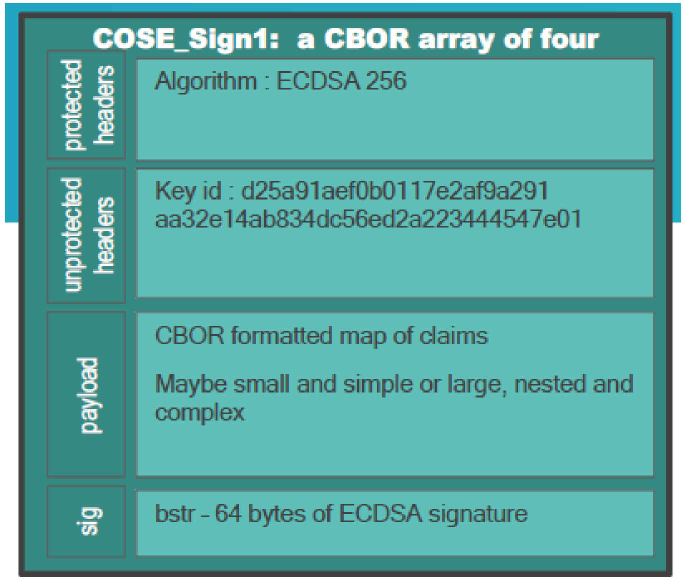
2. Meaning of Individual Claims
   - Interoperability between devices and servers from different vendors

3. Signing Format
   - Accommodate different schemes and algorithms

4. Encryption Format (optional)
   - Accommodate different algorithms

arm

COSE



CBOR Object Signing and Encryption ("COSE")

An IoT-oriented format for signing and/or encrypting a payload

Much simpler and more compact than PKCS #7, CMS and JOSE

COSE provides structuring of payload, algorithm identification, key identification and signature

COSE signed tokens are small, self-secured data blobs

Standard format (RFC 8152) allows use and development of standard / open source tools

T-COSE library

What is symmetric key based attestation?

- Device is provisoned with shared symmetric key (device and verifier).
- Symmetric key is used to generate a token authentiction tag, which ensures the token integrity and authenticity: HMAC tag
- The rest is more or less the same.

arm

What we gain with symmetric keys?

- Flash space
- Dropping asymmetric crypto algorithms from crypto service reduce its size significantly.
- TF-M Profile Small is addressing constrained devices, where image size really matters.
- HMAC based token authentication using hashing algorithm only and no asymmetric crypto algorithm.

arm

What we can lose with symmetric keys?

- Limited use cases and higher cost of the associated infrastructure for key management and operational complexities.
- In case of HMAC (due to the shared secrets) the DM or CM might need to run the verification service, while in the other case this can be done by a third party: cloud service provider.
- The usage of symmetric keys make the system more vulnerable to secret disclosure.
- Private keys are only stored on device, but symmetric keys must be known by both party: device and verifier.
- If the database with the symmetric keys becomes compromised, then all corresponding devices become untrusted.
- Since a centralized database of symmetric keys may need to be network connected, this can be considered to be a valuable target for attackers.

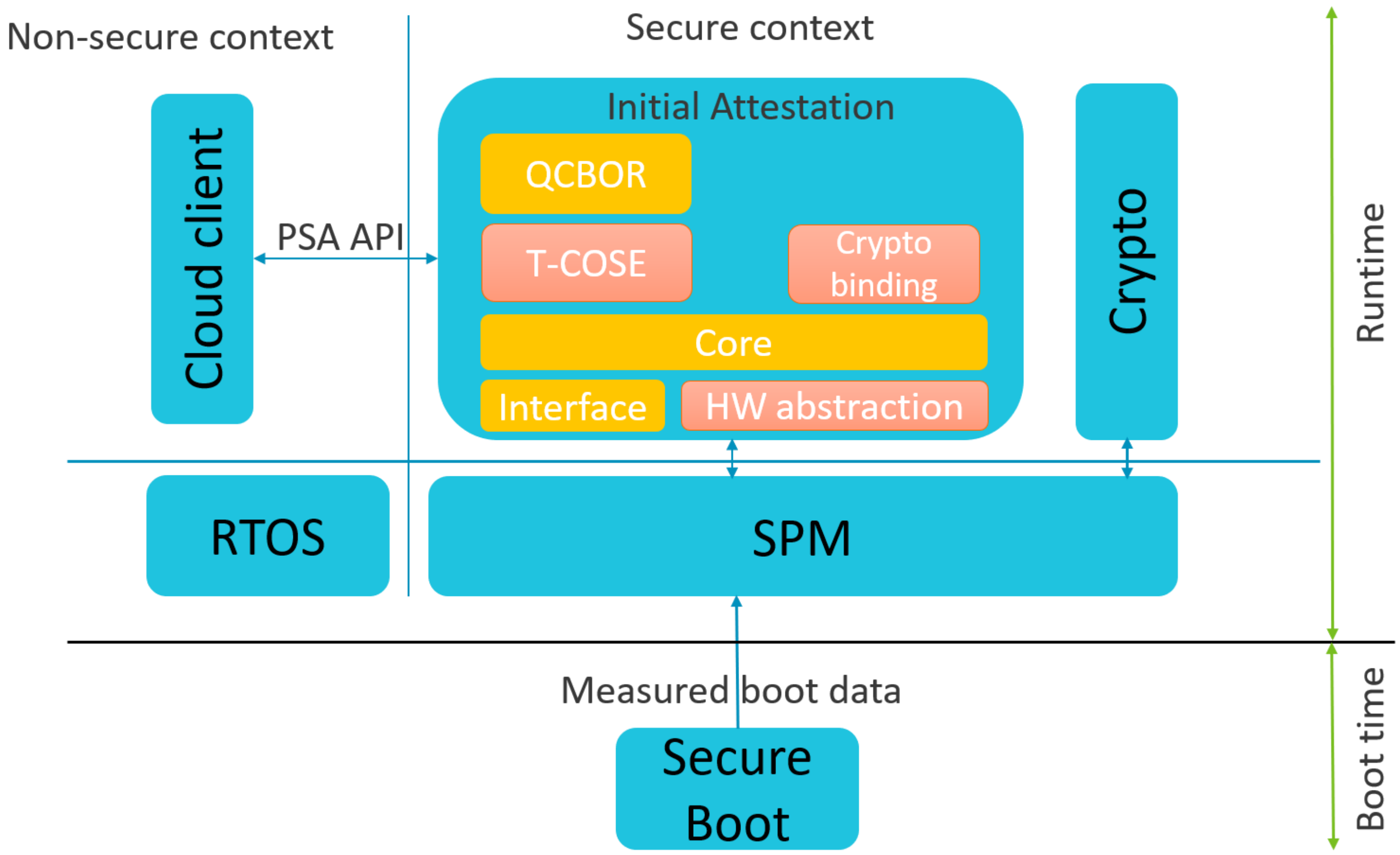
ECDSA vs. HMAC

	ECDSA	HMAC
Secret stored	Device	Device + verification database
Verification database	Public keys	Same symmetric key
Protection of the verification database	Integrity	Integrity + confidentality
Who can verify token?	Third party	CM or DM
Crypto algorithms	Hash + elliptic curve	Hash
Flash requirements	High	Low

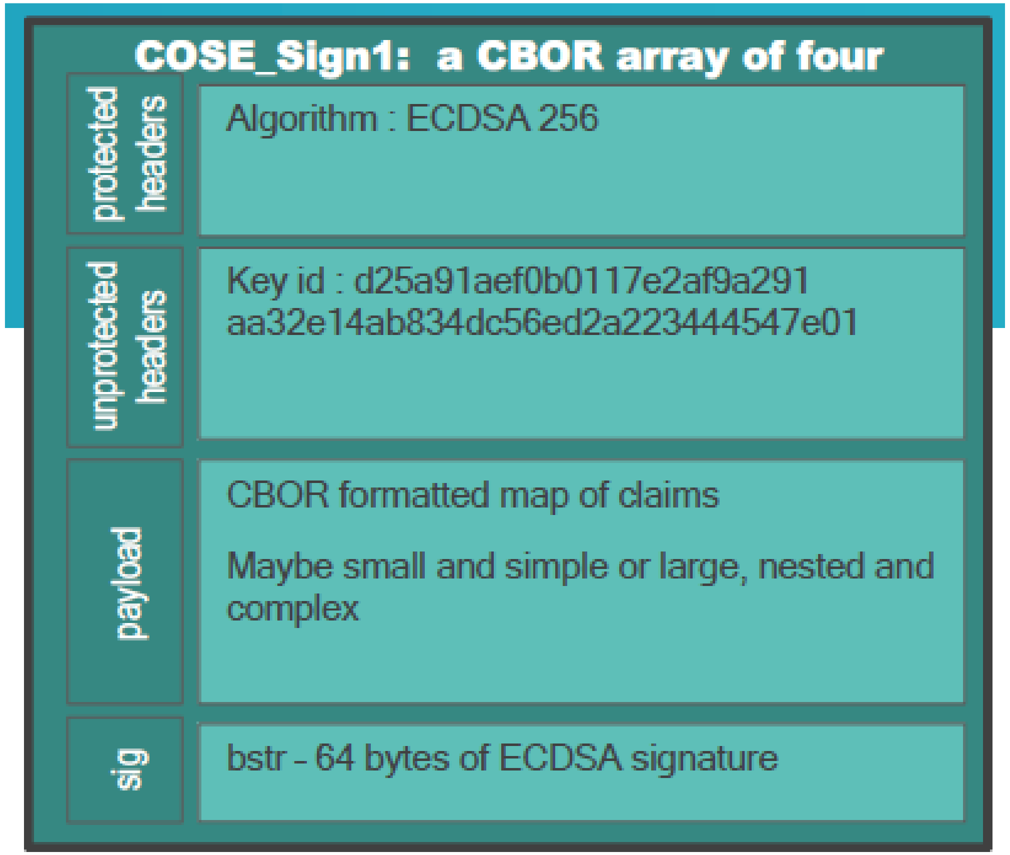
arm

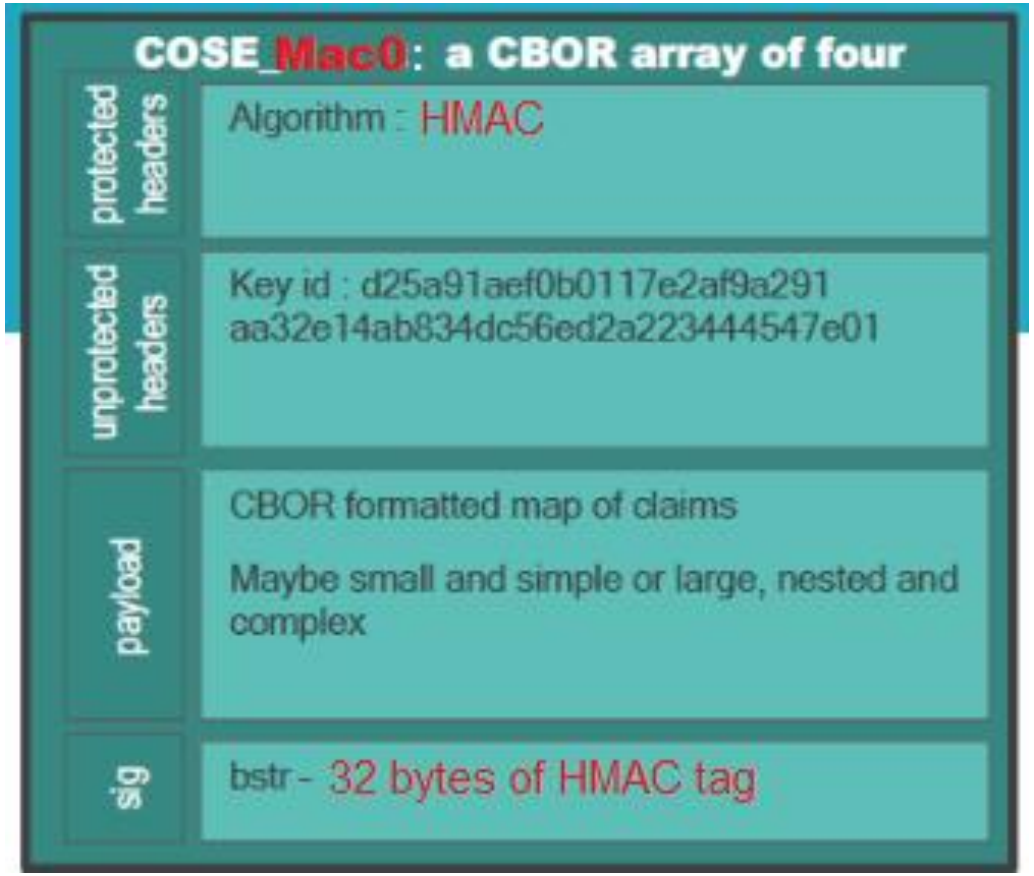
Affected SW components

- API does not change
- HMAC can be enabled by compile time switch



Difference in the token





More info

PSA Attestation API

TF-M Initial Attestation user guide

TF-M Initial Attestation code

Design proposals:

- Symmetric key based device attestation
- Comparison of asymmetric and symmetric key based device attestation

