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.
Attestation overview

- Device-unique cryptographic key is securely provisioned during manufacturing
- Verification key and HW ID is extracted and registered to database
- Firmware versions and their measurements 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
- Device 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(...)

Remote server
Secure - PRoT
Non-secure
HW

Remote server
Non-secure
Secure - PRoT
HW

Initial attestation request
Initial attestation key
Initial attestation

Object record C

Object record B

Object record A

Validating entity

Challenge

Initial attestation report

Response

Initial attestation request

Initial attestation

Attestation request

Object record A

Initial attestation

Sign

Attestation request

Initial attestation

Object record B

Object record C
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
**CBOR**

“Concise Binary Object Representation” (CBOR, [http://cbor.io](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
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 provisioned with shared symmetric key (device and verifier).
• Symmetric key is used to generate a token authentication tag, which ensures the token integrity and authenticity: HMAC tag
• The rest is more or less the same.
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.
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

<table>
<thead>
<tr>
<th>Comparison</th>
<th>ECDSA</th>
<th>HMAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secret stored</td>
<td>Device</td>
<td>Device + verification database</td>
</tr>
<tr>
<td>Verification database</td>
<td>Public keys</td>
<td>Same symmetric key</td>
</tr>
<tr>
<td>Protection of verification database</td>
<td>Integrity</td>
<td>Integrity + confidentiality</td>
</tr>
<tr>
<td>Who can verify token?</td>
<td>Third party</td>
<td>CM or DM</td>
</tr>
<tr>
<td>Crypto algorithms</td>
<td>Hash + elliptic curve</td>
<td>Hash</td>
</tr>
<tr>
<td>Flash requirements</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>

## Crypto Algorithms
- **Hash + elliptic curve** for **ECDSA**
- **Hash** for **HMAC**

## Flash Requirements
- **High** for **ECDSA**
- **Low** for **HMAC**
Affected SW components

- API does not change
- HMAC can be enabled by compile time switch
Difference in the token

**COSE_Sign1: a CBOR array of four**

- **protected headers**
  - Algorithm: ECDSA 256

- **unprotected headers**
  - Key id: d25a91aef0b0117e2af9a291a32e14ab834dc56ed2a22344547e01

- **payload**
  - CBOR formatted map of claims
    - Maybe small and simple or large, nested and complex

- **sig**
  - bstr - 64 bytes of ECDSA signature

**COSE_Mac0: a CBOR array of four**

- **protected headers**
  - Algorithm: HMAC

- **unprotected headers**
  - Key id: d25a91aef0b0117e2af9a291a32e14ab834dc56ed2a22344547e01

- **payload**
  - CBOR formatted map of claims
    - Maybe small and simple or large, nested and complex

- **sig**
  - bstr - 32 bytes of HMAC tag
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