EAT Proposals and Issues

IETF 106, Singapore, November 2019

Laurence Lundblade

When is EAT Done?

Should the following claims be added now or later?

Proposed: Attestation Result

- The output of the verifier can be an EAT
  - (even when input is YANG and comes from a TPM)
- Implicit claims can be made explicit
  - The verifier may know somethings about the device based on successful use of the attestation key, debug is disabled
  - A measurement input can be turned into a good / bad result
- A claim for certifications received by the device
  - Valuable when verifier is a neutral party that can cross-check

Proposed: Public key claim

FIDO and Android KeyStore both do *key attestation*. IoT on boarding often needs a key pair for use with DTLS.

A public key is included in an attestation so a relying party can know:

- The public key (and that the device has the private key)
- Key generation conditions (based on many already existing claims)
- Conditions governing key use (e.g., user authentication on device)

The work would be to standardize claims so standard EAT is closer to what FIDO, Android and IoT need. Supporting everything used by these is most likely too large and complex. New claims:

- Public key claim – reuse COSE and JOSE structures?
- Proof of possession of public key
- Some of the three-dozen characteristic Android uses

Proposed: Software Inventory Claim

For each software item on the device the following might be claimed:

- Name and / or file path
- Type – ROM, flashed execute in place, ELF, app…
- Version
- Vendor / Author
- Some measurement or authentication. Some or all of:
  - Hash of the file or code
  - Certificate that was used to verify it
  - Hash of certificated used to verify it
  - DN of certificate used to verify it

A likely structure is an array with one member for each SW item that is a map containing some of the above claims similar to ARM's PSA Token.

Proposed: Software Inventory Claim

Motivation:

- ARM PSA Token supports
- Cisco publishes a known-good-values list of its SW
- Seems of general interest

Challenges:

- Doing it in an OS-neutral way

Proposed: Measurement / Integrity Claim

Support several styles of measurement and integrity check:

- TPM-style: a series of hashes are taken at *boot time* and reported to verifier
- Run time integrity: hashes taken during *run time* and reported to verifier
- Integrity check result: run time hashes are *evaluated in attester* and only true/false is reported to verifier (attester has to have known-good-values).

Proposed: Measurement / Integrity Claim

Motivation:

- Widely in use in TPM-based architecture
- Samsung TIMA (as an example of a product)
- TPMs can’t support run time integrity check

Challenges:

- Finding platform / architecture neutrality
- Finding a small enough set of measurement schemes
- Finding common ground between TPM and non-TPM worlds

Pull Requests (PRs) and Open Issues

PR updating `oemid` claim

PR has corrected references to IEEE

- Claim is an IEEE MA-L, MA-M, MA-S, a company prefix for a block of MAC addresses of length 24, 28 or 36 bits
- Claim may also be an IEEE CID, not a MAC address, but from the same space (doesn't collide)

PR has new encoding

- Specially encoded as:
  - Hexadecimal text in JSON for easy viewing and as is common with MAC addresses
    - AC-DE-48-23 or AC:DE:48:23
  - A byte string in CBOR for compactness

PR to add **`Nonce`** claim

- Previously the **`jti`** and **`cti`** claims were used for the nonce from the relying party
- **`jti`** and **`cti`** are *token identifiers* from JWT and CWT and may be generated locally.
- Now **`nonce`** claim is used
  - Already defined in IANA JWT registry
  - EAT defines nonce for CWT, as it is not in the IANA CWT registry

PR for debug states

- Previously array of four independent Booleans:

```
boot_state_type = [
    secure_boot_enabled=> bool,
    debug_disabled=> bool,
    debug_disabled_since_boot=> bool,
    debug_permanent_disable=> bool,
    debug_full_permanent_disable=> bool
]
```

- Now similar, but an enumerated type with five states

```
debug_disable_level = (
    not_disabled: 0,
    disabled: 1,                    May have been enabled earlier
    disabled_since_boot: 2,
    permanent_disable: 3,           Only the manufacturer can enable
    full_permanent_disable: 4       Not even the manufacturer can enable
)
```

Discussion on debug states

```
debug_disable_level = (
    not_disabled: 0,
    disabled: 1,                    May have been enabled earlier
    disabled_since_boot: 2,
    permanent_disable: 3,           Only the manufacturer can enable
    full_permanent_disable: 4       Not even the manufacturer can enable
)
```

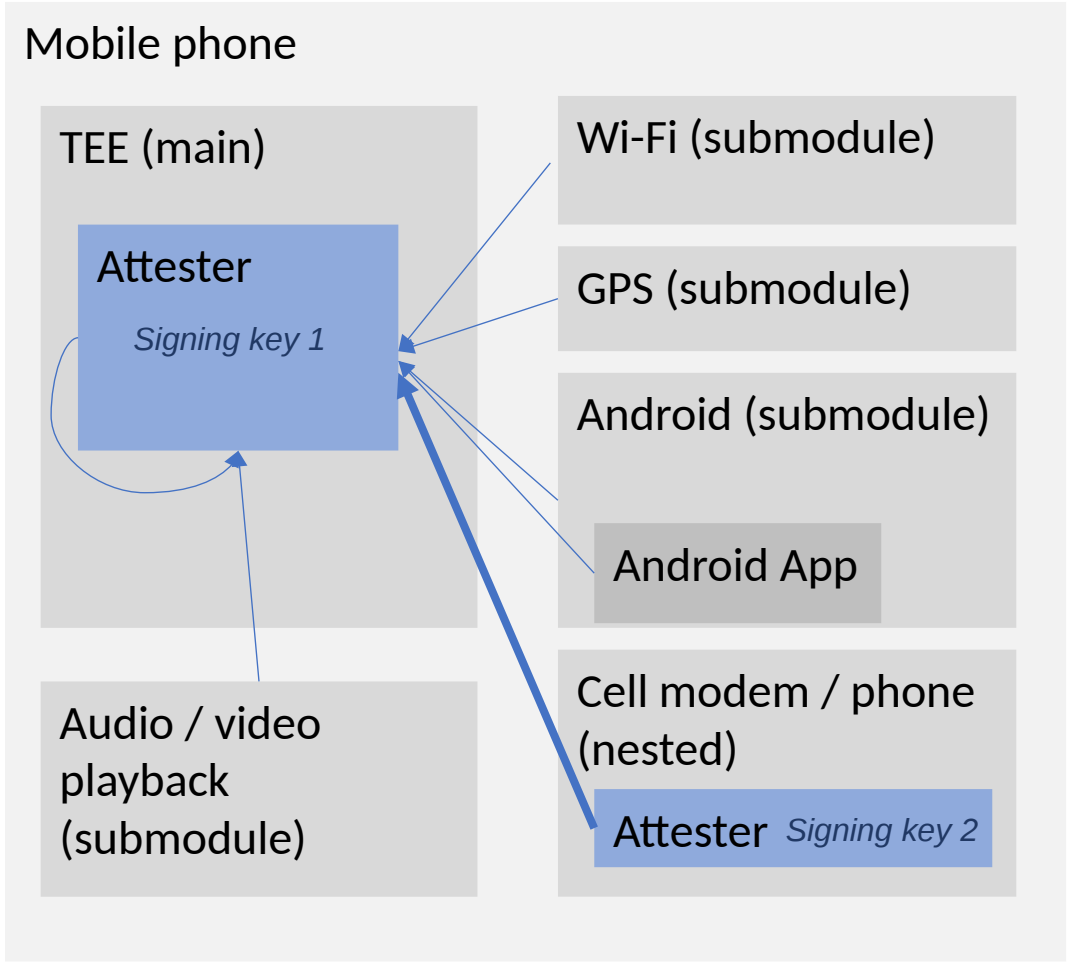
This applies to HW or broad system SW debug facilities, not to in-process debuggers like gdb.

With the new non-inheritance submods, this is not inherited. Each subsystem must indicate its debug state.

When a debug system has access to or effects multiple submods, each submod must still report its stated individually.

PR – New `submods` structure

Mobile phone example; submods all on internal bus



- Each submodule feeds claims to the attester
  - The chip / system architecture allows the Attester to know which claims come from which submodule
- Each submodule has
  - A string name
  - Claims…
  - Indicator of attachment strength
- Claims are NOT inherited
  - Each submodule has its boot and debug stated, OEM ID, Version…
- Two types
  - No signing key: feeds individual claims to attester
  - With a singing key / subordinate attester: feeds a fully serialized and signed EAT to attester
  - (Possibly a third type that feeds a hash of serialized claims)

Description of changes in the PR

- Unifies signed and unsigned submodules; both now under **`submods`**
  - The **`submods`** part of a token is a map with one submodule per entry
  - **`submod_name`** replaced by putting the name in the **`submods`** map label / key
  - The **`nested_eat`** claim is removed
  - A signed submodule, a signed encoded token (formerly a nested_eat) is a map entry in **`submods`**
- New **`submod_attachment`** claim is added
  - Described how the submodule is attached to the attester
  - Enumerated: unspecified, device internal, PCB internal, chip internal

Abbreviated Submods Example

```
{
   / nonce /                  7:h'948f8860d13a463e8e',
   / UEID /                   8:h'0198f50a4ff6c05861c8860d13a638ea4fe2f',
   / time stamp (iat) /       6:1526542894,
   / seclevel /              11:3, / secure restricted OS /

   / submods / 17:
      {
         / 1st submod, an Android Application / "App Foo": {
           / nonce /                  7:h'948f8860d13a463e8e',
           / seclevel /      11:1, / unrestricted /
           / submod_attachment / 24: 4 / chip internal /
           / app data /  -70000:'text string'
         },
         / 2nd submod, A nested EAT from a cell modem /  "Cell Modem": {
           / eat /         16:61( 18(
              / an embedded EAT / [ /...COSE_Sign1 bytes with payload.../ ]
                         ))
         }
         / 3rd submod, information about Linux Android / "Linux Android": {
            / nonce /             7:h'948f8860d13a463e8e',
            / seclevel /          11:1, / unrestricted /
            / submod_attachment / 24: 4 / chip internal /
            / custom - release / -80000:'8.0.0',
            / custom - version / -80001:'4.9.51+'
         }
      }
}
```

Security Considerations

<add Giri’s text here>

UEID Size Discussion

UEID sizing is not the same as for IP addresses

- UEIDs must never be reassigned or reused over time or space
- Devices NOT IP connected; are bus connected, Bluetooth connected, serial port connected…
- There are likely to be very large databases of devices in IoT backend services, but not IP addresses

People	Devices/person	Resulting database size	Scenario likelihood	Discussion
10 billion	100	trillion	Highly realistic and fully expected	128 bits is enough
10 billion	100,000	quadrillion	Edge of what we might expect	128 bits may be marginal
100 billion	1,000,000	100 quadrillion	Speculative – devices per mammal, nanobots…	Need a least 192 bits

Options:

1. Permanent limit at 128 bits
2. Require 128 bits now, allow for 256 bits
3. Require 256 bits now

Should randomly generated UEID be 128 bits or 256 bits?

Database Size

People	Devices/person	subsystems / device	Database portion of population	Resulting database size
10 billion	100	10	.1	trillion
10 billion	100,000	10	.1	quadrillion
100 billion	1,000,000	10	.1	100 quadrillion

Probability of collision in one instance of database calculated by birthday attack

Database size	128 bits	192 bits	256 bits
trillion	2 * 10^-15	8 * 10^-35	5 * 10^-55
quadrillion	2 * 10^-09	8 * 10^-29	5 * 10^-49
100 quadrillion	2 * 10^-05	8 * 10^-25	5 * 10^-45

Time to collision assuming 10% of database changes per year

Database size	128 bits	192 bits	256 bits
trillion	60,000 years	10^{24} years	10^{44} years
quadrillion	8 seconds	10^{14} years	10^{34} years
100 quadrillion	8 microseconds	10^{11} years	10^{31} years

Claims Characteristics PR, slide 1

General advice on claim design

- Interoperability and Relying Party Orientation
  - Design claims so relying parties can understand what they mean
- OS and Technology Neutral
  - Not specific to operating system, hardware, programing language, manufacturer, sub industry
  - E.g., don’t orient to TEE, TPM, Unix, mobile phones, Javascript…
- Security Level Neutral
  - Claims that are good for high security environments (TPMs, secure elements) and low security environments (user mode apps).
- Reuse of Extant Data Formats
  - Don’t reinvent when existing structures can be re used; re use expertise
  - Various approaches to encoding (translate to CDDL, take as is…)

Claims Characteristics PR, slide 2

General advice on claim design

- Proprietary Claims
  - Considering the forgoing, proprietary claims are explicitly allowed

- Profiles
  - Separate documents that may
    - Make some claims mandatory
    - Prohibit others
    - Define new claims
    - Narrow meaning of existing claims

Other claims that are in process or proposed

- Origination
- Profile
- Boot Seed