Proposal for TEEP Attestation Structure

David M. Wheeler

Intel Corporation

Capabilities of Attestation Mechanism (Issue #17)

- From current draft

```
Attestation is the process through which one entity (an attestor)
presents a series of claims to another entity (a verifier), and
provides sufficient proof that the claims are true. Different
verifiers may have different standards for attestation proofs and not
all attestations are acceptable to every verifier. TEEP attestations
are based upon the use of an asymmetric key pair under the control of
the TEE to create digital signatures across a well-defined claim set.
```

- Attestor = the TEE
- Verifier = the TAM
- Sufficient Proof = Digital Signature using TEE Private Key
- Claims = ???

Claims

- Desired Requirement for Claims
  - Well-Structured
    - Easy to parse, Unambiguous
    - Implies a language for claims
  - Minimally Self-Contained
    - Should not require significant outside material for interpretation for security policy decisions
  - Extensible
    - We won’t think of all the claims necessary, so we should plan for extensions
  - Support for Proprietary Signatures
    - Existing platforms won’t be compliant, so we need to incorporate proprietary attestations that the TAM will have to interpret

Complexities of Platform Environments

- Stand Alone HW-Based TEEs
  - Intel SGX
    - Attestation includes HW/uCode versions & HW-fuse key
- Composed HW/SW Based TEEs
  - Arm Trustzone
    - Attestation requires proof of Secure Boot (version & measurement), trusted firmware (TFW), and TEE environment (version & measurement), and TEE Key linked to TFW
- Virtual Machine / Platform-Based 'TEEs'
  - Link GPU / FPGA / ASIC to TEE environment for
    - AI Algorithm / AI Training Set usage (e.g rental, IP licensing)
    - FaaS (e.g. payment services, media processing, multi-party computation)
  - Usage: Cloud Service Providers (CSP), IoT at the Edge

Claim Language Options

- SUIT Manifest
  - Is self-contained & Extensible
    - CDDL is a good option for definition
  - Not Unambiguous
    - Structure is not designed for attestation
    - Requires forced redefinition of elements to insert necessary claim components

Not Recommended

- EAT Token
  - Is well-structured & self-contained
    - Good start on dictionary of claims; recognition of need for extensions
  - Extensibility is left as an unmanaged subclaim (not ideal)

Recommended

- Merge EAT & SUIT Manifest
  - Use CDDL to define elements
  - Utilize EAT concept for overall structure & beginning dictionary of claims
  - Proposal planned for RATS

Recommended

Summary Recommendation

- Utilize a CHOICE based structure in TEEP
  - Combination of CDDL, EAT Token and Proprietary Attestation
- Proposed CDDL follows this slide
- Work with RATS / EATS to merge CDDL with EAT Token

An Attestation

- Choice of Three Options
  - TEEP Attestation Token
  - EAT Attestation Token
  - Proprietary Attestation Token

```
CDDL
AttestationToken = {
   tokenType:     uint,
   token:     TEEPAttestationToken \\
              EATToken \\
              ProprietaryToken
}
ProprietaryToken = {
    pTokenType:    uint,
    token:         bstr
}
```

A TEEP Attestation

- Claim Type – unique identifier
- Claim Instance – discriminator to further identify type or instance
- Claim Version - string
- Claim Proof – binary
    - Measurement, hash

- The *proof* signature is to be signed by the TEE key
    - Other enclosed signatures may be signed by other RoTs

CDDL

```
TEEPAttestationToken = {
   hdr:      AttestationHdr,
   *claims: ClaimSet,
   proof:    Signature
}
```

Attestation Header

The Header is used to determine the type of attestation this token represents, in order to help find the correct verifier, and contains basic information to associate this attestation with a particular protocol exchange (freshness), device (DevID) or entity (ori).

- Version: Major.Minor
- Attestation type – registered number for type of known attestations, with one reserved for proprietary
- Freshness of the attestation
  - Basic Timestamp
  - Nonce – received from requestor
  - Transaction ID – rec'd from requestor
  - Counter – number of ticks since reboot
- Device Identity
  - Optional to allow for use of EPID or other anonymous signing algs
- Originator
  - TEE, TPM, Kernel, or TA
  - Hash of Public Key
  - Certificate

CDDL

```
AttestationHdr = (
   ver:   Version,
   atype: uint,
   ?fresh: Timestamp // TxID //
           Nonce-Challenge // Counter
   ?DevID: EAT-UEID // Device-Model-No
    ori:   EAT-Origination // PKHash //
           Certificate
)
Version = (
   maj: uint,
   min: uint
)
Device-Model-No = (
    oem: OEM-Identifier
    model: CPUSVN // bstr // tstr
)
```

A ClaimSet

- Claim Dictionary – reference to registry of claim formats contained in this claim set (e.g. Profile Defn)
- Claim Purpose – the purpose for this set of claims – e.g. Device identification, TEE or TA Identification, Boot Claims, Debug claims, etc.
- Claims – array of 1 or more claims that are grouped or signed

- Purpose for signed claims is to allow different claims to be signed by different Root-of-Trust Keys in a platform or composed TEE

CDDL

```
ClaimSet = (
   ?dic: tstr,
   ?purpose: tstr,
   +claims:  GroupedClaims //
             SignedClaims
)
GroupedClaims = (
   +claims:   Claim,
)
; allows nesting signed claims
SignedClaims = (
    +claims:  Claim,
    sig:      Signature
)
```

A Claim

- Claim Type – unique identifier, or measurement type
- Claim Instance – discriminator to further identify type or instance, or measurement description
- Claim Version – string
- Origin – indication of where this element was sourced/origin; ex. for SW/FW this is the entity that signed the component; hash of the PK
- Claim Proof – binary
  - Measurement Value, or hash

CDDL

```
Claim = (
   uid:      tstr,
   ?inst:    tstr,
   ?version: tstr,
   ?origin:  tstr // bstr
   proof:    bstr
)
```

Measurement items are mapped to draft-Tschofenig-rats-psa-token-01.txt

A Signature

- Who signed this Attestation
- Algorithm for signature
- Reference or direct Pub Key used to verify signature (e.g. cert)
- signature

CDDL

```
Signature = (
   attestor:  DeviceId //
              UID // PKHash //
               cert // tstr,
   alg: AlgorithmID,
   key: tstr // bstr,
   sig: bstr
)
```