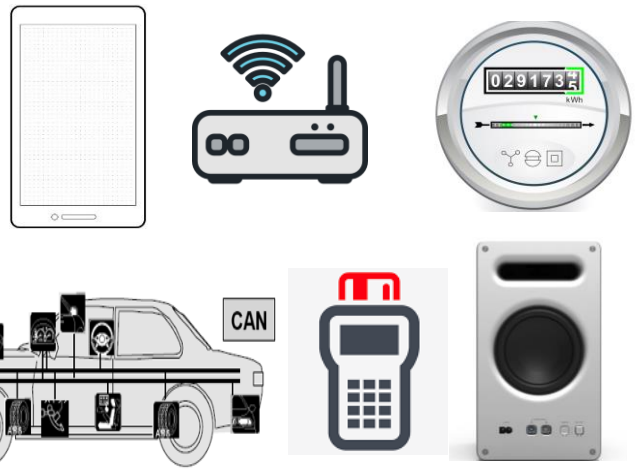
# Attestation (RATS/EAT) Overview

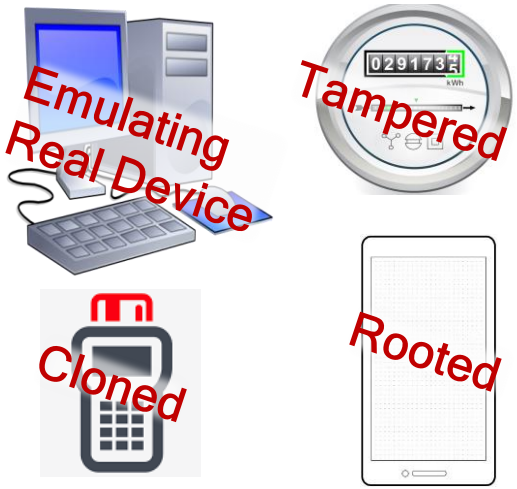
Laurence Lundblade

February 2020

Good Devices




Bad Devices




# Entity Attestation Token

- Chip & device manufacturer
- Device ID (e.g. serial number)
- Boot state, debug state…
- Firmware, OS & app names and versions
- Geographic location
- Measurement, rooting & malware detection…

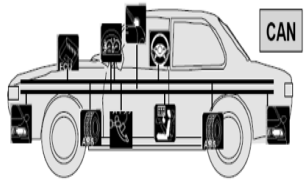
**All Are Optional**

Cryptographically secured by signing

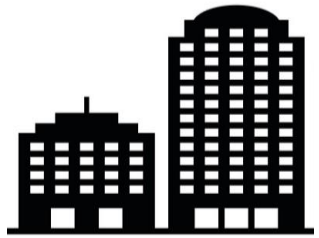
Banking risk engine

IoT backend

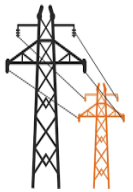
Network infrastructure




Car components



Enterprise auth risk engine



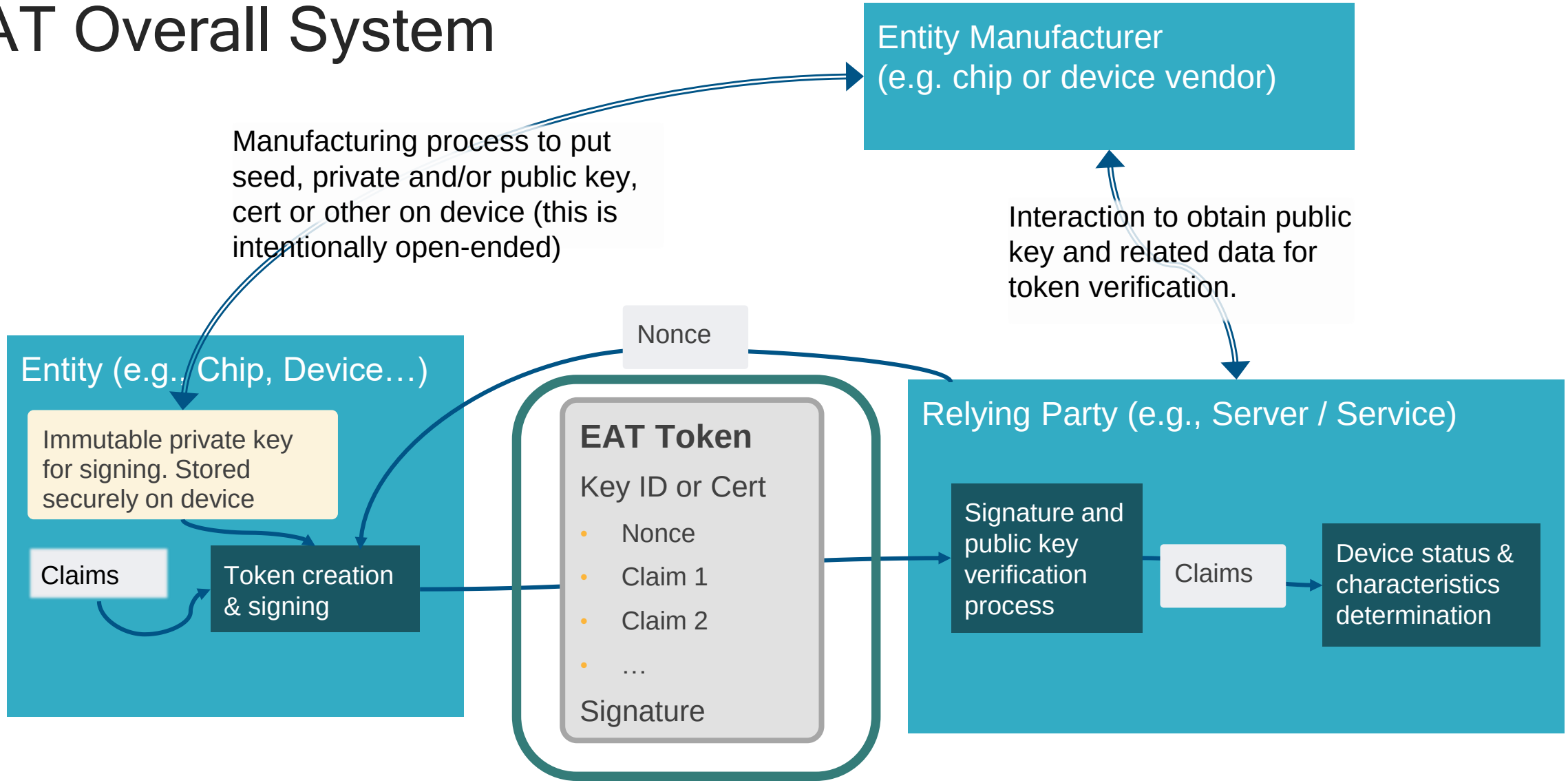
Electric company

# EAT Initial Set of Claims

| Claim | Description |
|---|---|
| UEID | Identify a particular individual device, similar to a serial number |
| OEM ID | Identify the manufacturer of the device |
| Boot and debug state | Is secure/trusted/authenticated boot turned on? Is debug disabled? |
| Geographic location | GPS coordinates, speed, altitude |
| Security level | Rich OS, TEE, secure element… |
| Nonce | Token freshness |
| Origination | Identifies authority that can verify the token |
| Time stamp | Time and / or age of the token |
| Submodules | How to deal with claims from different subcomponents of a module. For example, the TEE and Rich OS are separate submodules. |
| Nested tokens | Putting one EAT inside another as a way of handling subcomponents |

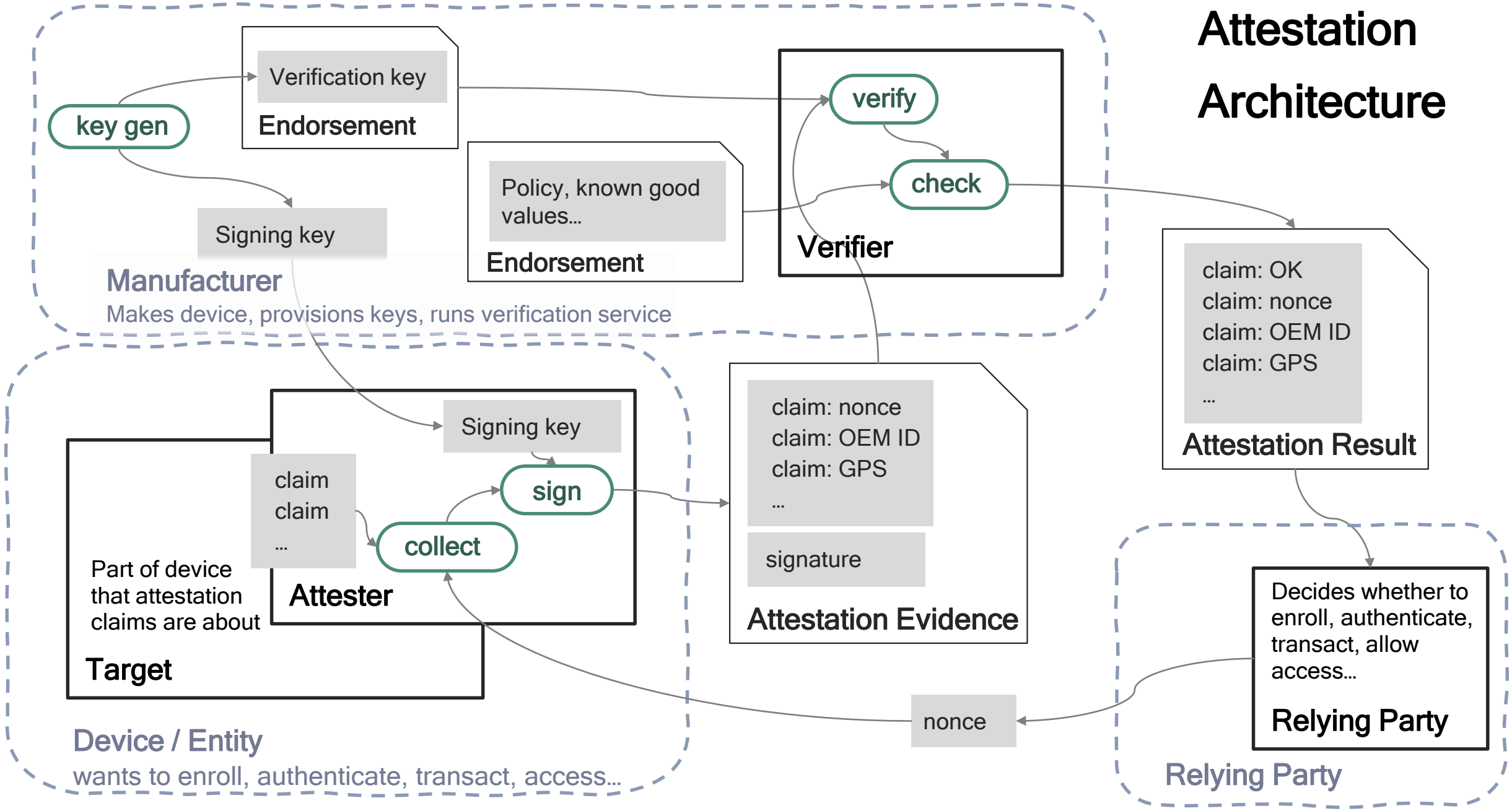
Intended only as initial set. Expansion should include SW components, measurement, public keys (similar to Android attestation) and other.

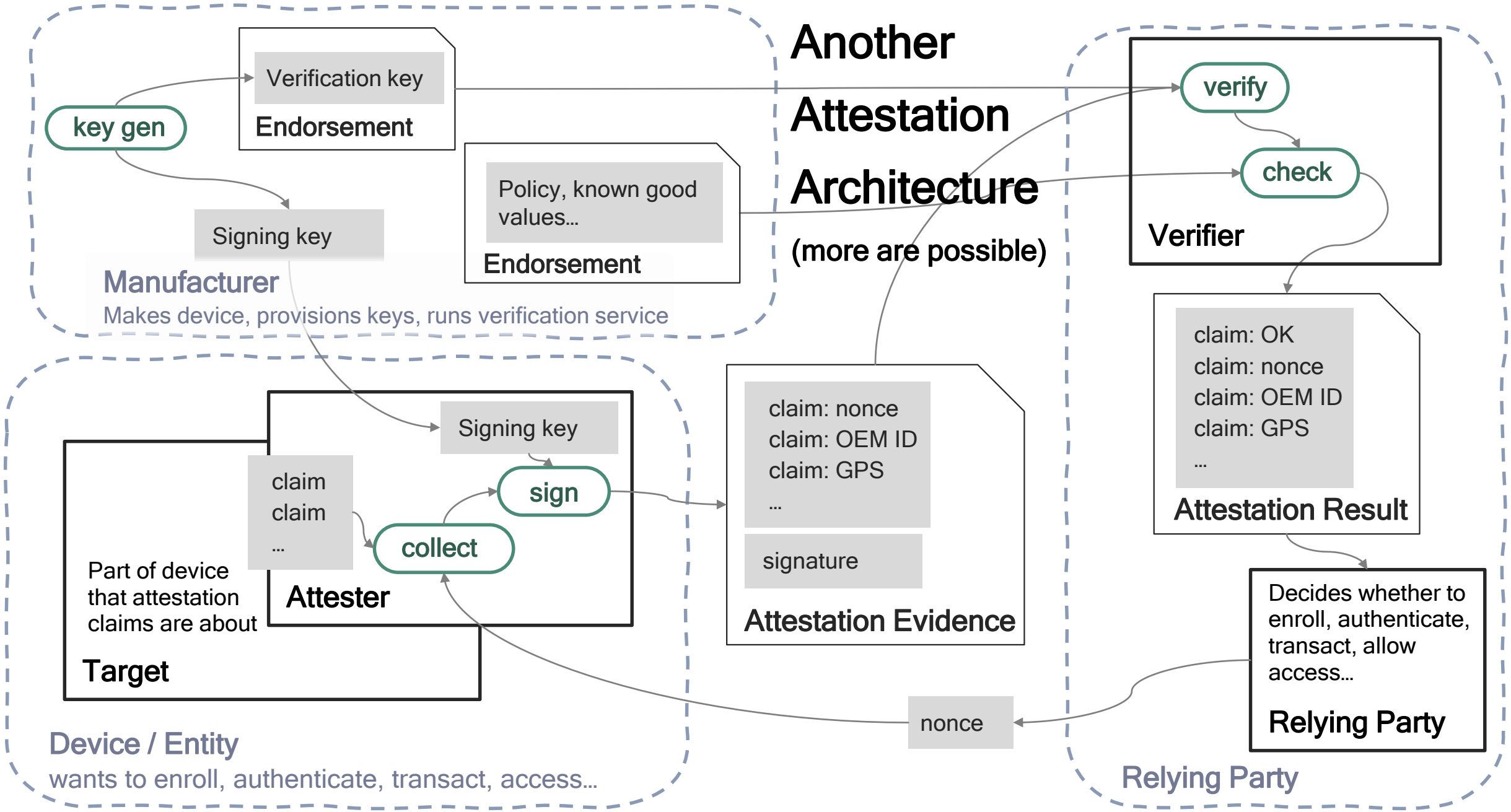
# EAT Overall System




**EAT Target for standardization**

# Attestation Architecture



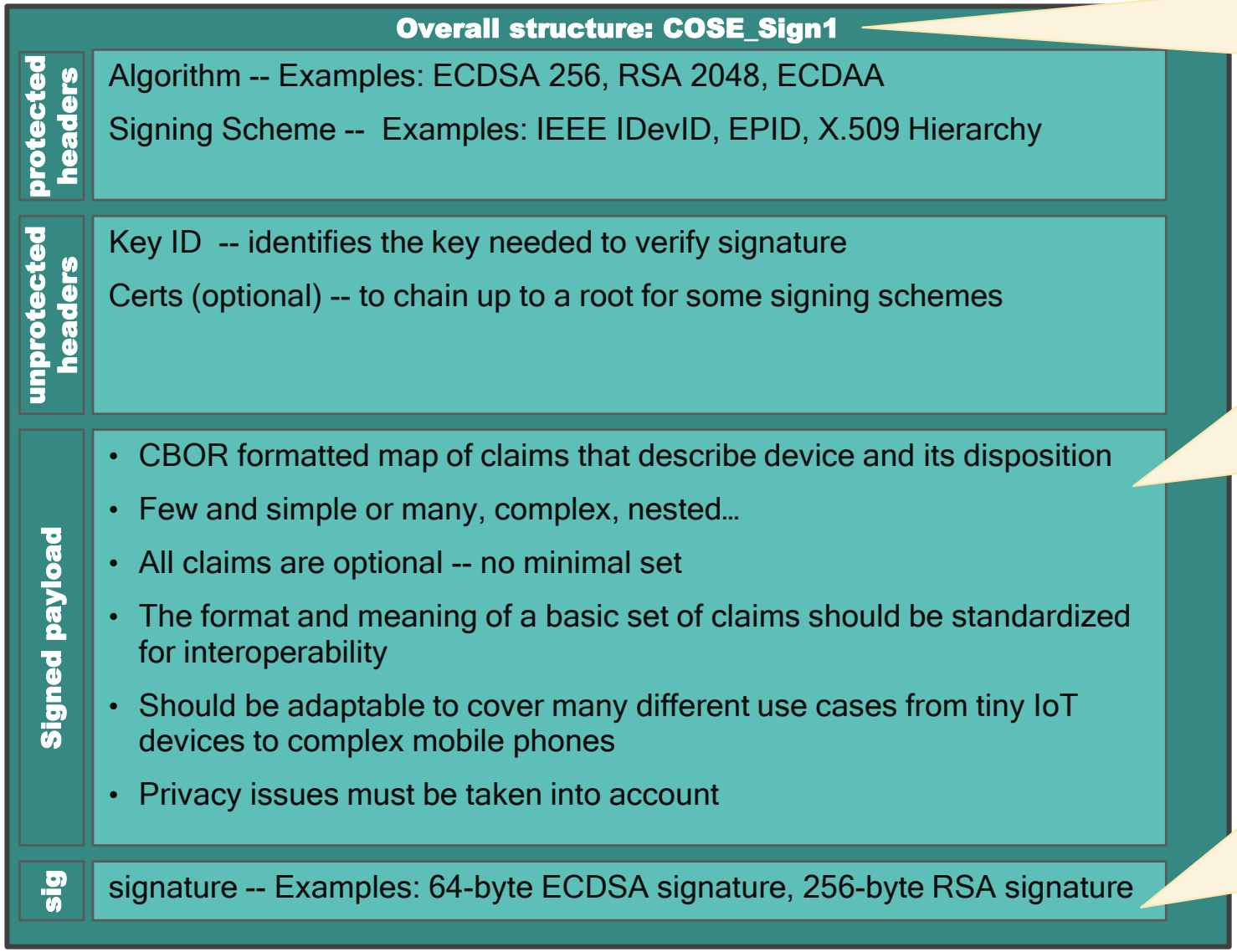

Another
Attestation
Architecture
(more are possible)
key gen
Verification key
Endorsement
Signing key
Policy, known good values…
Endorsement
Manufacturer
Makes device, provisions keys, runs verification service
verify
check
Verifier
claim: OK
claim: nonce
claim: OEM ID
claim: GPS
…
Attestation Result
Signing key
sign
claim
claim
…
collect
Part of device that attestation claims are about
Attester
Target
claim: nonce
claim: OEM ID
claim: GPS
…
signature
Attestation Evidence
Decides whether to enroll, authenticate, transact, allow access…
Relying Party
nonce
Device / Entity
wants to enroll, authenticate, transact, access…
Relying Party

# Primary Standardization Goal is Semantic Interoperability of Claims

- Main types of claims to standardize:
  - Device Identity
  - Measurement
  - Device boot, debug and configuration state
  - Measurement and run time integrity checks
  - Geographic location
  - Device SW and HW versions
  - Public key created on the device – Keystore, IoT and FIDO use cases
- Claims should be generally applicable:
  - Not specific to TPM, TrustZone, SGX, Secure Element…
  - Not require any particular level of device security
    - Works with high-security device like Secure Elements and TPMs and low-security devices with nothing special at all.

# EAT Format (basically CWT)

draft-mandyam-eat-00




- COSE format for signing
- Small message size for IoT
- Allows for varying signing algorithms, carries headers, sets overall format

- CBOR format for claims
- Small message size for IoT
- Labelling of claims
- Very flexible data types for all kinds of different claims.
- Translates to JSON

- Signature proves device and claims (critical)
- Accommodate different end-end signing schemes because of device manufacturing issues
- Privacy requirements also drive variance in signing schemes

# Example Token

COSE binary ~130 bytes including sig

COSE ECDSA signing overhead is about 87 bytes: 23 for headers and structure, 64 bytes for ECDSA sig

JSON text ~500 bytes including a JOSE sig

CBOR diagnostic representation of binary data of full signed token

```
[
  / protected / << {
    / alg / 1: -7 / ECDSA 256 /
  } >>,
  / unprotected / {
    / kid / 4: h'4173796d6d65747269634543445341323536'
  },
  / payload / << {
    / UEID / 8: h'5427c1ff28d23fbad1f29c4c7c6a55',
    / secure boot enabled / 13: true
    / debug disabled / 15: true
    / integrity / -81000: {
       / status / -81001: true
       / timestamp / 21: 1444064944,
    },
    / location / 18: {
       / lat  / 19: 32.9024843386,
       / long / 20: -117.192956976
    },
 } >>,
  / signature / h'5427c1ff28d23fbad1f29c4c7c6a555e601d6fa29f9179bc3d7438bacaca5acd08c8
                 d4d4f96131680c429a01f85951ecee743a52b9b63632c57209120e1c9e30'
]
```

Payload Translated to JSON

- Integer labels mapped to strings
- Binary data base 64 encoded
- Floating point numbers turned into strings

```
{
   "UEID" : "k8if9d98Mk979077L38Uw34kKFRHJgd18f==",
   "secureBoot" : true,
   "debugDisable" : true,

   "integrity": {
       "status": true,
       "timestamp": "2015-10-5T05:09:04Z",
   },
   "location": {
       "lat": "32.9024843386",
       "long": "-117.192956976",
   },
}
```

# COSE Signing Scheme Flexibility

- Many standard algorithms already supported
  - RSA, ECDSA and Edwards-Curve Signing (public key)
  - HMAC and AES-based MACs (symmetric key)
- Extensible for future algorithms
  - IANA registry for algorithms exists today
- Extensible for special case schemes
  - Proprietary simple HMACs schemes, perhaps HW based
  - Possibly Intel EPID
  - (non-standard algorithms will of course be less interoperable)

# Privacy

- Entity Attestation Tokens are intended for many use cases with varying privacy requirements
  - Some will be simple with only 2 or 3 claims, others may have 100 claims
  - Simple, single-use IoT devices, have fewer privacy issues and may be able to include claims that complex devices like Android phones cannot

- Options for handling privacy
  - Omit privacy-violating claims
  - Redesign claims especially to work with privacy regulation
  - Obtain user permission to include claims that would otherwise be privacy-violating

- Some signing schemes will be privacy-preserving (e.g. group key, ECDAA) and some will not

# Detailed Claims Description

# Nonce

## Basic Claim

A unique string from the relying party

Included in token to prevent replay attacks

# Universal Entity ID (UEID)

## Basic Claim defined in EAT draft

### Identify an individual manufactured entity, device, chip, box…

- Like a serial number, but not necessarily sequential
- NOT a model number, device type or class of device
- Universally and globally unique across all devices from all manufacturers without any qualifier.
- Permanent, not reprogrammable
- Not intended for direct use by humans

### Several types of binary byte strings defined:

- Type 1 – 128 to 256-bit random number (e.g., a GUID)
- Type 2 – IEEE EUI (similar to or same as MAC addresses registered by company by IEEE)
- Type 3 – IMEI (typical mobile phone serial number)
- Types 4,5,6 – IEEE EUI-48, 60 and 64

The relying party, receiver or consumer, MUST treat this as a completely opaque identifier

# OEM ID

## Basic Claim defined in EAT draft

This identifies the manufacturer of the entity

- IEEE OUIs are used here since IEEE provides a global unique registry of companies
- This is commonly the first part of a MAC address
- Perhaps a GUID can also be used to avoid IEEE fees and entanglements

Identifies a device of a certain brand, a chip from a particular manufacturer, etc.

By using submodules (defined later), a single token can identify the OEM of the chip(s), module(s) and final consumer product.

# Boot and Debug State

Basic Claim defined in EAT draft

Allow relying party to understand if the device is fully secured and under control of the OEM

**Secure Boot Enabled Boolean**

- Indicates only SW authorized by the OEM is running

**Debug Enablement Status**

- Mostly relates to HW-based debug facilities including RMA diagnostics

| | |
|---|---|
| debugDisabled | Debug is currently disabled, but may have been previously enabled |
| debugDisabledSinceBoot | Debug has not been enabled in this boot cycle, but may have been enabled in previous boot cycles |
| debugPermanentDisable | Debug can only be enabled by the OEM |
| debugFullPermanentDisable | It is not possible to enable debug |

# Token Time Stamps

## Basic Claim defined in EAT draft

| | |
|---|---|
| Time stamp | Epoch-based time indicating when the token was created.<br>Optional (as all claims are) since some entities do not have a clock |
| Age | Number of seconds since token or data was generated<br>Useful only if token data is cached or pre-generated some time before token is sent |
| Uptime | Number of seconds since the device booted |

# Geographic Location -- WGS84 Coordinate System

Basic Claim defined in EAT draft

All claims are optional

All can be either integer or float

| | |
|---|---|
| Latitude | |
| Longitude | |
| Altitude | |
| Accuracy | Accuracy of latitude and longitude in meters |
| Altitude accuracy | Accuracy of altitude in meters |
| Heading | 0 to 360 |
| Speed | Meters/second |

# Security Level

## Basic Claim defined in EAT draft

Rough characterization of the overall security of the entity implementation

Primarily characterizes the protection of the attestation signing key

Only rough characterization is possible as this can be very subjective. The relying party must be aware of this and may want to rely other claims instead.

| | |
|---|---|
| Unrestricted | The implementor has made some attempt to protect the attestation key<br>Example: Linux, Windows, MacOS kernel or system process |
| Restricted | Uses a subsystem, but not one that is security-oriented.<br>Example: Wi-Fi subsystem, IoT device |
| Secure restricted | Uses a security-oriented restricted operating environment<br>Defend against large-scale network based attacks<br>Examples: TEE, Virtualization Based Security, Intel SGX |
| Hardware | Defends against physical or electrical attacks<br>Examples: secure elements, smartcards, TPMs |

# Origination

Basic Claim defined in EAT draft

Identifies the part of a device originating the token

May tie back to manufacturer and/or URL for verification of the token

(This needs refinement)

# HW Version

Basic Claim defined in PSA draft

International Article Number, IAN-13, a 13-digit number

Superset of 12-digit UPC (standard barcode)

Used by some chip vendors to version IC layout sent to the fab

General broad product identification use

# Boot Seed

## Basic Claim defined in PSA draft

A large random number regenerated every time the entity boot cycles

Allows relying party to tell if the device has rebooted since the last token was received

# Profile Definition

Basic Claim defined in PSA draft

URI / string identifier of profile document describing the token and use case in more detail

May include:

- Standardized claims allowed or used for this profile
  - Restrictions on these standard claims
- Definitions of new / custom (not standard) claims
- Claims that are mandatory / optional
- Submodule structure for profile
- Signing scheme

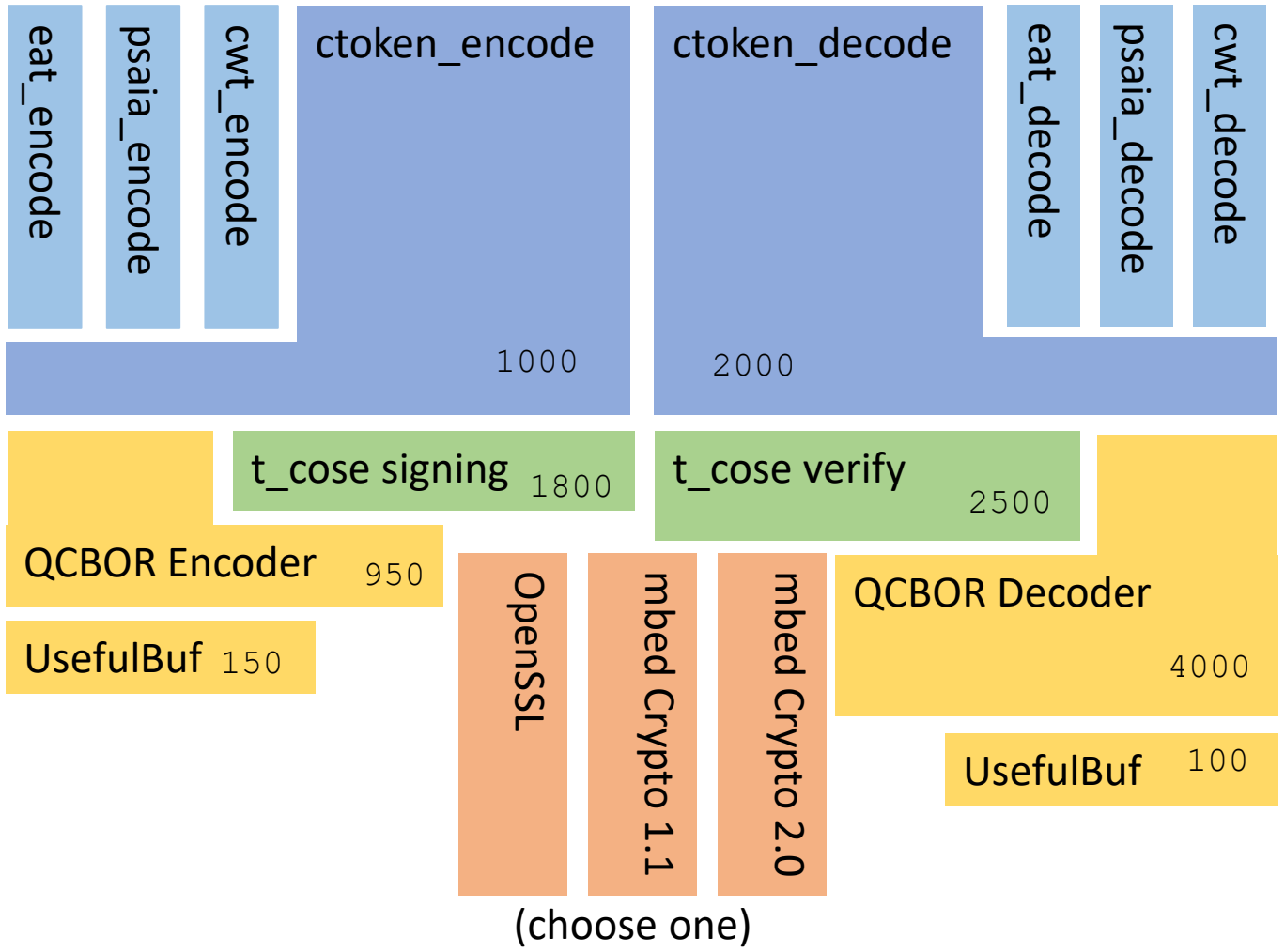
# The Other More Complex Claims

The following claims areas were not discussed in this presentation:

- SW Components
- Measurement and Integrity Checking
- Public keys and their characteristics (e.g. Android Keystore)
- Submodules and Nesting

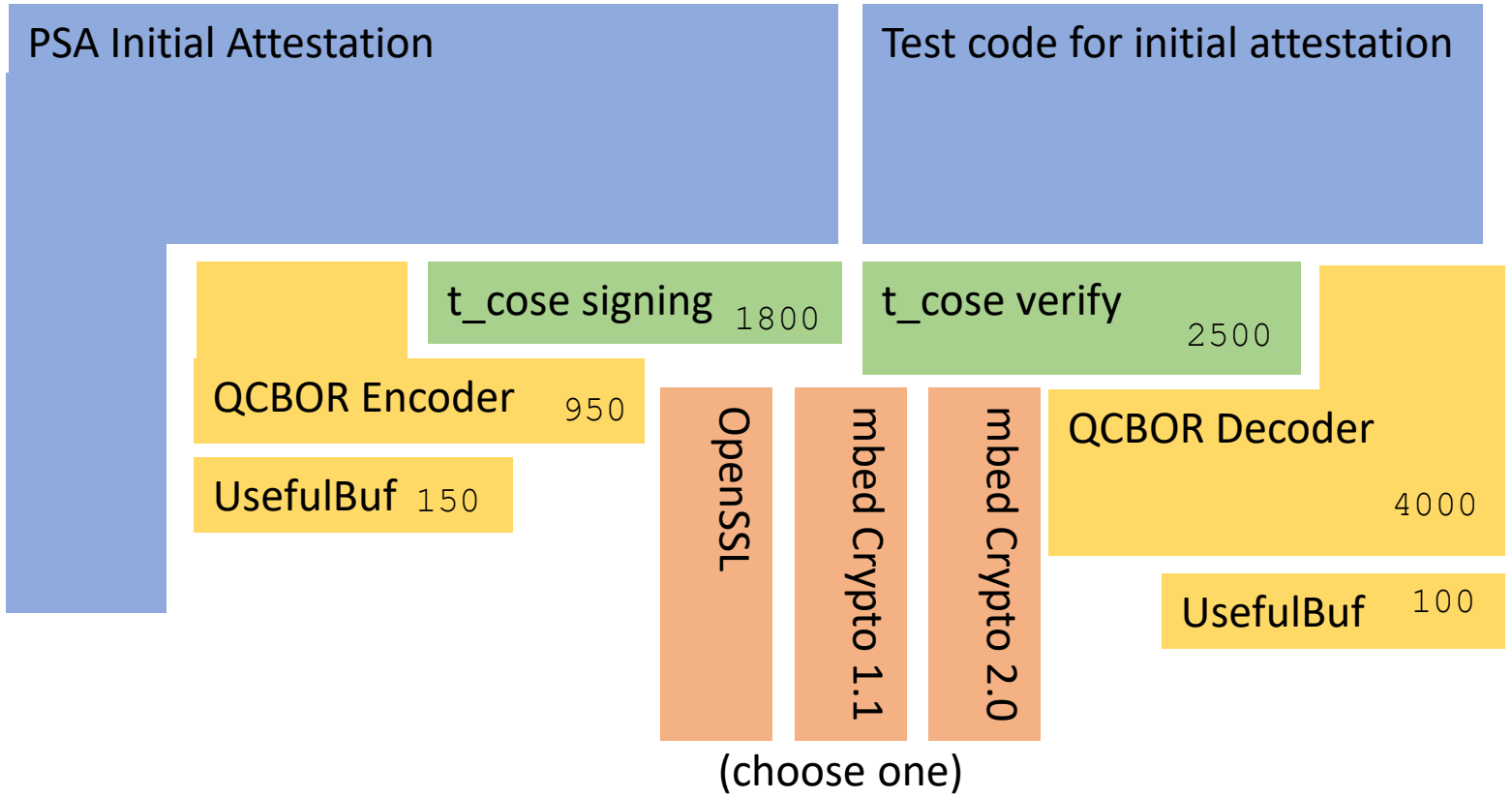
# QCBOR, t_cose, ctoken SW Stack

# Cake Diagram for ctoken / t_cose / QCBOR




- Encode and decode are largely separate except for crypto
- Sizes are for 64-bit x86 and ***very approximate***
- Sizes vary by compiler
- Sizes vary, particularly for encoding side, depending on the data types of the claims

# Cake Diagram for PSA initial attestation / t_cose / QCBOR



# QCBOR

- Full CBOR encoder / decoder written at Qualcomm and open-sourced
  - Now maintained by Laurence
- Easy to port in new environments
  - Dependencies: <stdint.h>, <stddef.h>, <stdbool.h> and <string.h>
- No malloc. Caller fully controls memory management
- Comprehensive automated test suite
- Secure coding style to avoid buffer overruns and vulnerabilities
- Stable for over a year, integrated with ARM TF-M

# t_cose

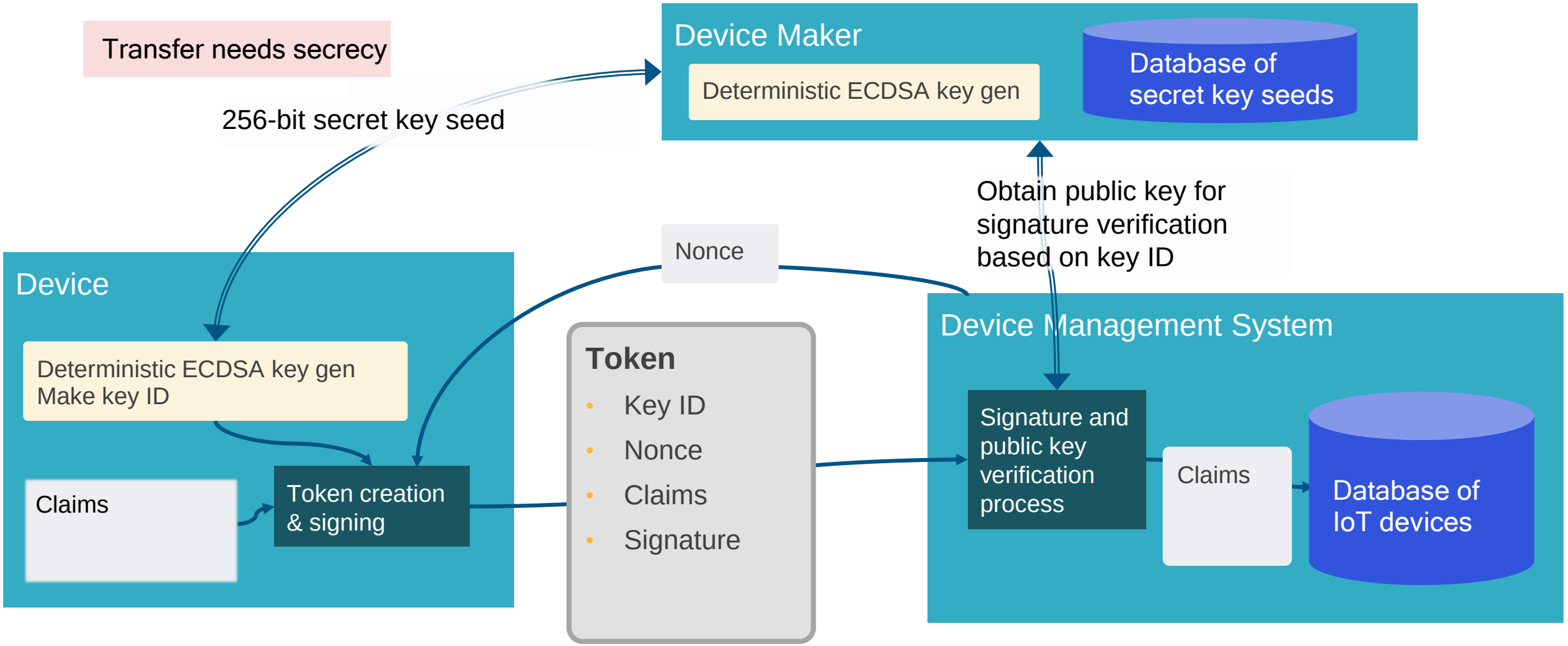
- COSE implementation of signing and verification targeted at embedded CWT & EAT
  - No encryption support, at least not for a while
- Dependencies:
  - <stdint.h>, <stddef.h>, <stdbool.h> and <string.h>
  - QCBOR
  - A crypto library for SHA hashes and ECDSA
    - OpenSSL (uses malloc)
    - PSA Crypto (either 1.1 or 2.0)
    - Future libraries via adaptor layer
- No malloc. Caller fully controls memory management
- Only one copy of the payload / token needed in memory to sign or verify
- Comprehensive test coverage
- Largely stable, integrated with ARM TF-M

# ctoken

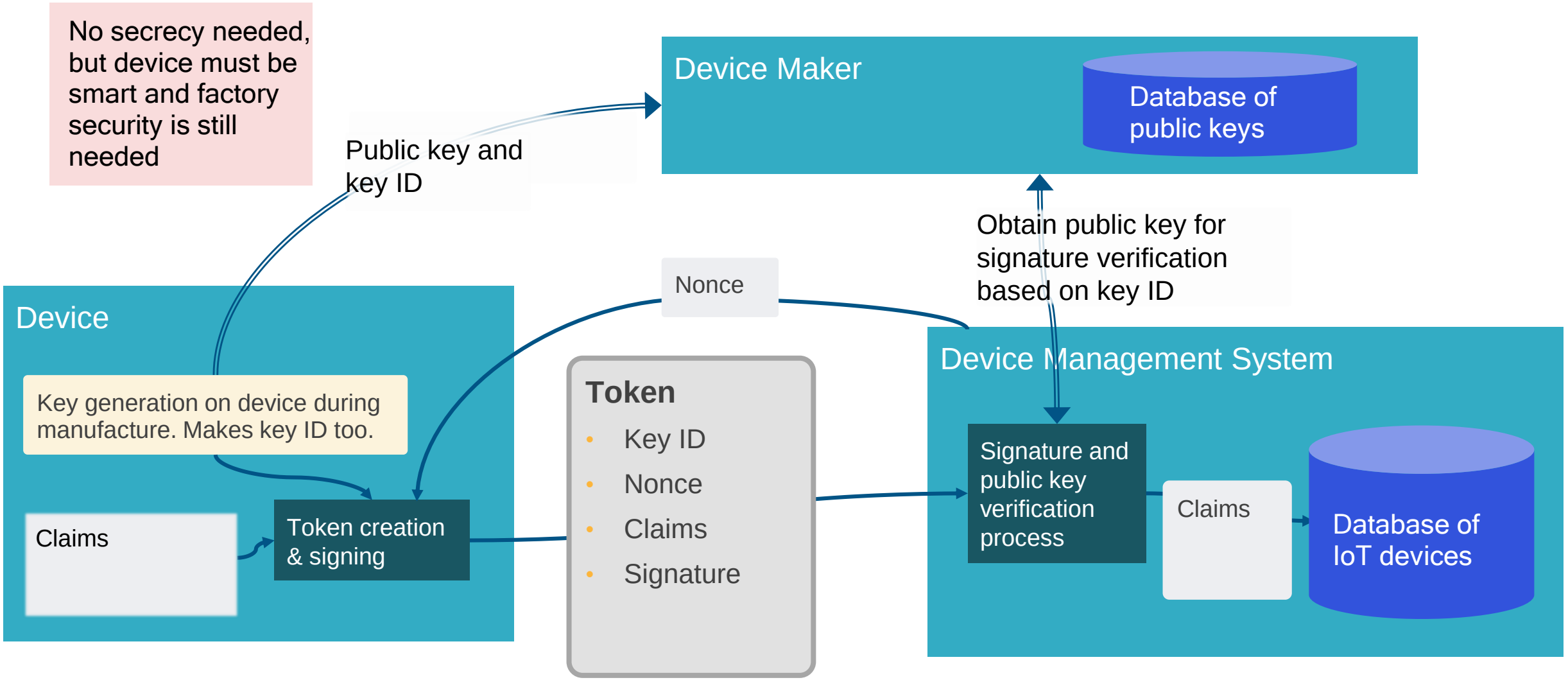
- Implements EAT, CWT and PSA Attestation
  - Flexible to add claims, combine claims from different standards
  - Only encoding / decoding of claims since claim values come from the OS or other
- Dependencies: t_cose and its dependencies
- No malloc. Caller fully controls memory management
- Only one copy of the payload / token needed in memory to sign or verify
- Derived and generalized from ARM PSA Attestation
  - The TF-M attestation API is different, higher level and include all claim generation

# Key Setup

# ECDSA key setup based on 256-bit secret seed



# ECDSA key setup generating key on device



# ECDSA key setup generating key outside of device