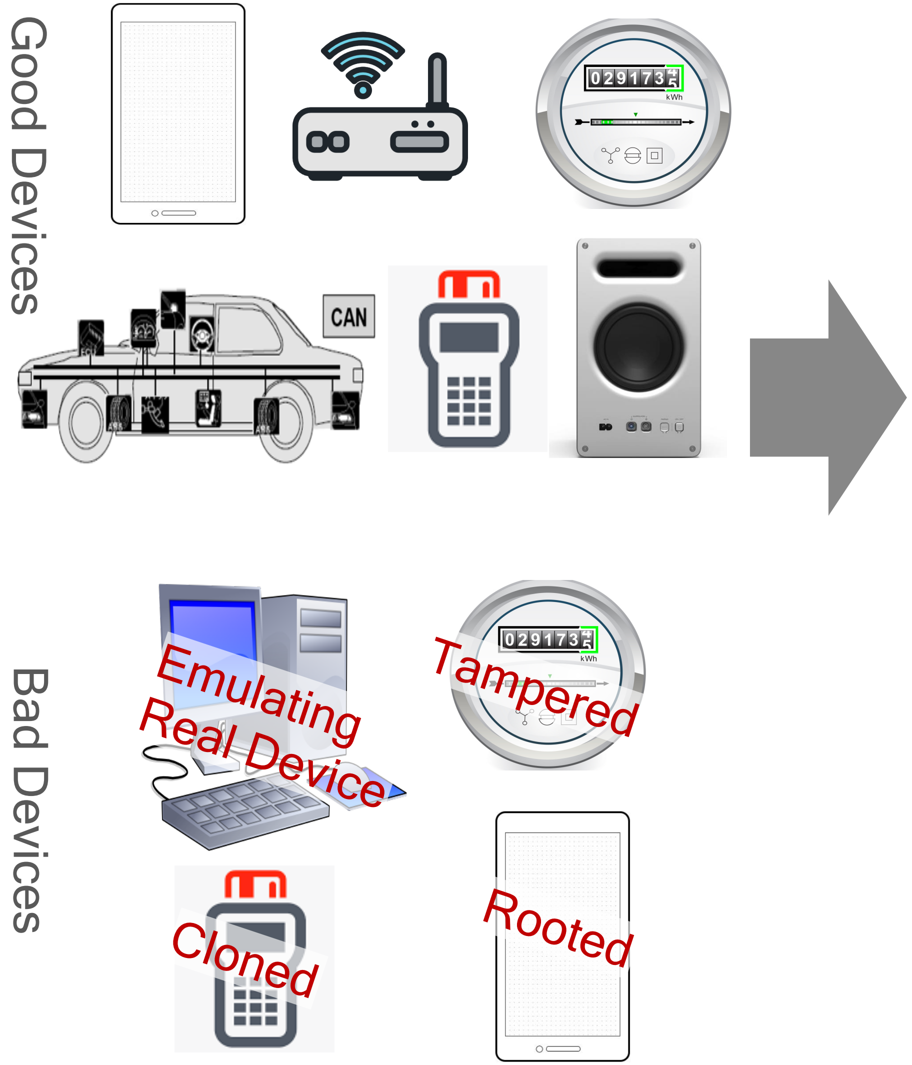
EAT
Entity Attestation Token

Laurence Lundblade

June 2021

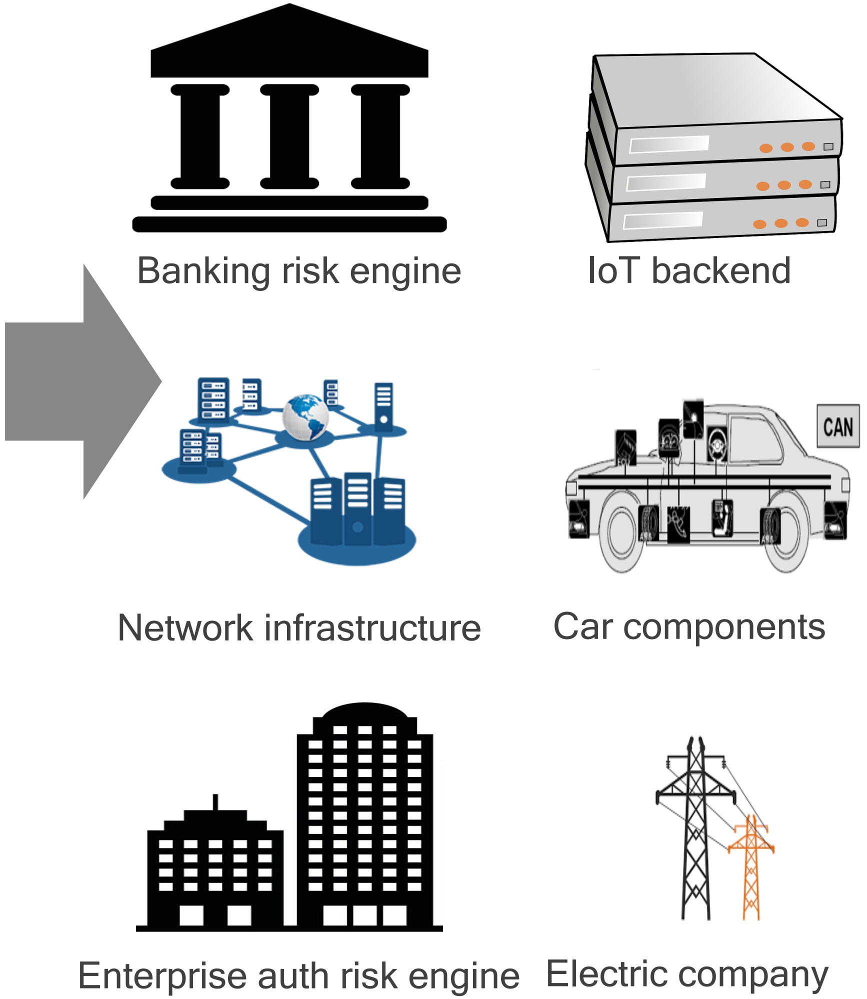


Entity Attestation Token

- 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

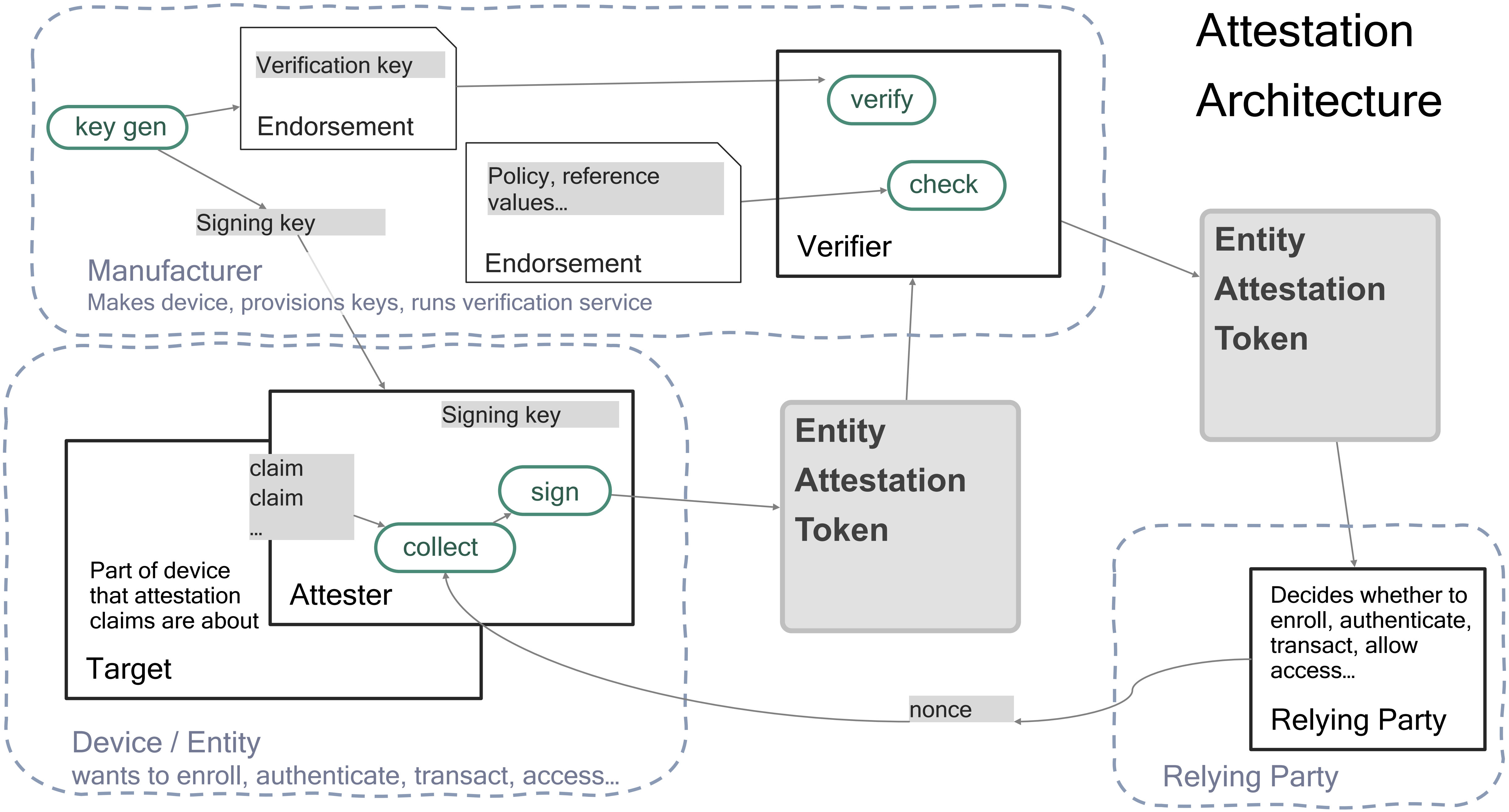
Cryptographically secured by signing



Overview of EAT Claims

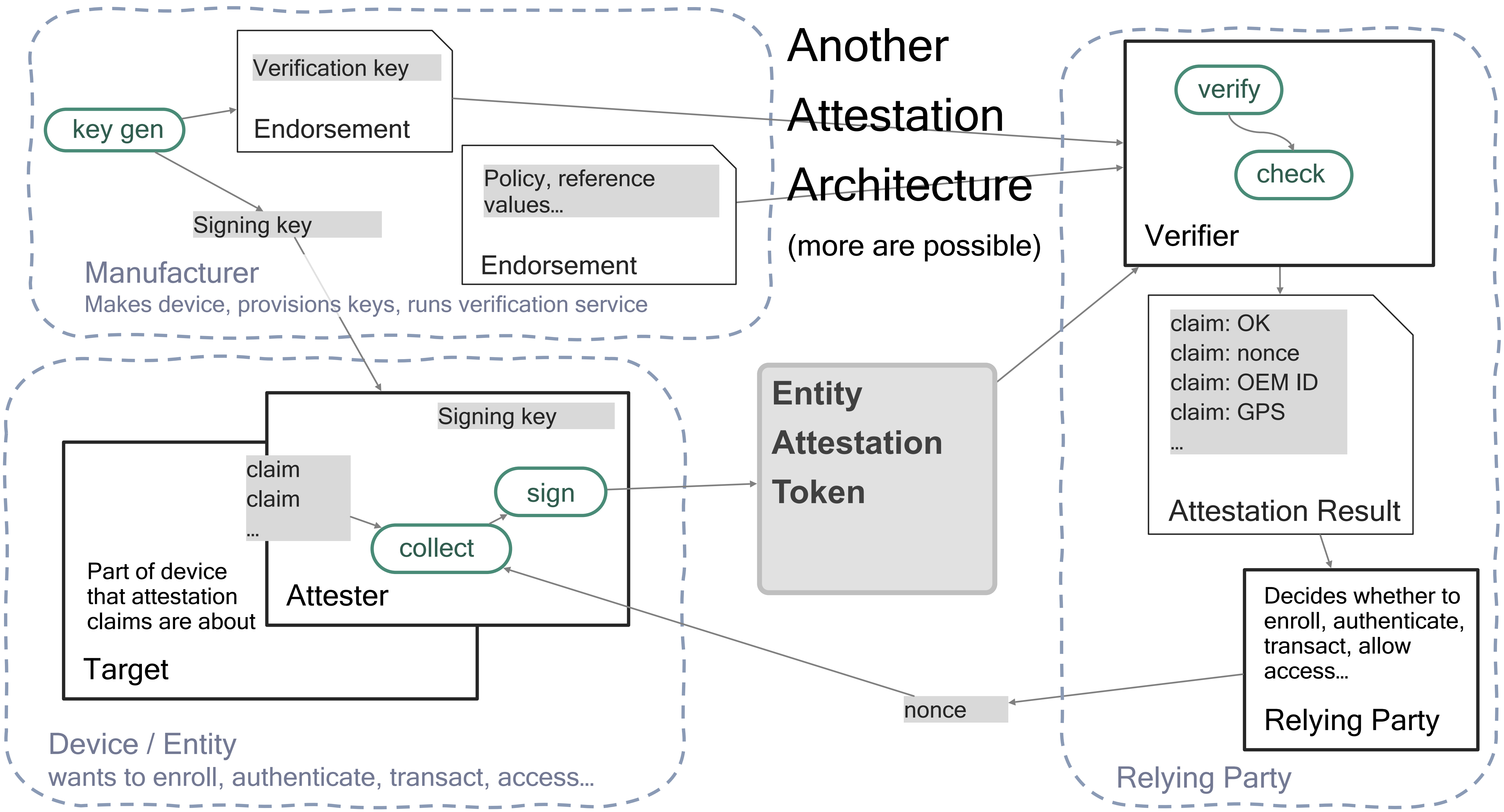
Claim	Description
UEID, SUEID	Identify a particular individual device, similar to a serial number
OEM ID	Identify the manufacturer of the device
Boot and debug state	Is secure/trusted/authenticated boot turned on? Is debug disabled?
Geographic location	GPS coordinates, speed, altitude
Security level	Rich OS, TEE, secure element…
Nonce	Token freshness
SW measurements	Hashes of SW components can be reported to verifier Measurements can also be verified on the device with only success/fail reported
SW manifests	Manifests of the SW installed on the system from SW supplier
Submodules	Subsets of claims from different subcomponents of a module. For example, the TEE and Rich OS can be separate submodules.
Nested tokens	Putting one EAT inside another as a way of handling subcomponents
Further, custom and proprietary claims	EAT claims can be registered with IANA, special space allocated for proprietary claims

Attestation Architecture



Another Attestation Architecture

(more are possible)



EAT Format – CWT with more claims

draft-mandyam-eat-10

Overall structure: COSE_Sign1

protected headers

Algorithm -- Examples: ECDSA 256, RSA 2048, ECDAA

unprotected headers

Key ID -- identifies the key needed to verify signature

Certs (optional) -- to chain up to a root for some signing schemes

Signed payload

- CBOR formatted map of claims that describe device and its disposition
- Few and simple or many, complex, nested…
- All claims are optional -- no minimal set
- Privacy issues must be taken into account

sig

signature -- Examples: 64-byte ECDSA signature, 256-byte RSA signature

---

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

---

- 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

EAT in JWT/JSON format

- CDDL is used to defined the claims for both JSON and CBOR
- A signed EAT can be either JWT or CWT format
- Translation from one to another is generally possible
- A JWT can even be nested in a CWT or vice versa, though this might not be the best idea

UCCS Format – Unprotected CWT Claims Sets

- This is a CWT without the COSE signing
- Useful if security is provided by other means such as TLS
- Separate internet draft: draft-birkholz-rats-uccs-01

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'5427c1ff28d23fbad1f29c4c7c6a555e601d6
                    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 are allowed, but will of course be less interoperable)

Privacy

- EATs 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
  - No specification for DAA yet; hopefully it will come

Detailed Claims Description

General notes about claims

- All claims are optional in the general standard
- Use cases can define profiles that make some claims mandatory, prohibit claims, …
- The CWT and JWT IANA claims registries are reused. New claims can be registered.
- Private and proprietary claims are allowed as per the CWT and JWT conventions

Nonce

- 8 to 64 byte binary value
- Multiple nonces are allowed for multi-stage validation and consumption
  - Both relying party and verifier should produce and check a nonce

UEID and SUEID

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)

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

SUEID – Semi-permanent

- UEID is like IDevID and SUEID is like LDevID
- SUEIDs can be created on device life-cycle events, IDevIDs cannot change
- Multiple SUEDs are allowed
- SUEID is the same format as UEID

Privacy

- Use case address privacy as needed
  - Don't use these IDs
  - Privacy proxy
  - User permission
  - …

OEMID

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

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.

Hardware Version Number

Three version numbers

- Chip
- Board
- Device

All three may be present

Reuses version scheme from CoSWID

- Versions are text strings
- A enumerated integer indicates their format and sorting order
- Adds EAN-13 (standard bar code) to types registered for CoSWID

Security Level

Rough indication of the security environment for the signing key and the construction of the claims

- **Unrestricted** — There is some expectation that implementor will protect the attestation signing keys at this level. Otherwise the EAT provides no meaningful security assurances.
- **Restricted** — Entities at this level should not be general-purpose operating environments that host features such as app download systems, web browsers and complex productivity applications. It is akin to the Secure Restricted level (see below) without the security orientation. Examples include a Wi-Fi subsystem, an IoT camera, or sensor device.
- **Secure Restricted** — Entities at this level must meet the criteria defined by FIDO Allowed Restricted Operating Environments [FIDO.AROE]. Examples include TEE's and schemes using virtualization-based security. Like the FIDO security goal, security at this level is aimed at defending well against large-scale network / remote attacks against the device
- **Hardware** — Entities at this level must include substantial defense against physical or electrical attacks against the device itself. It is assumed any potential attacker has captured the device and can disassemble it. Example include TPMs and Secure Elements

Secure Boot

Boolean value where true indicates the firmware and OS are under control of the OEM identified by the OEMID claim

Debug System Enablement

This is oriented to system/HW debug facilities like JTAG or RMA diagnostics, but can be applied to any debug system

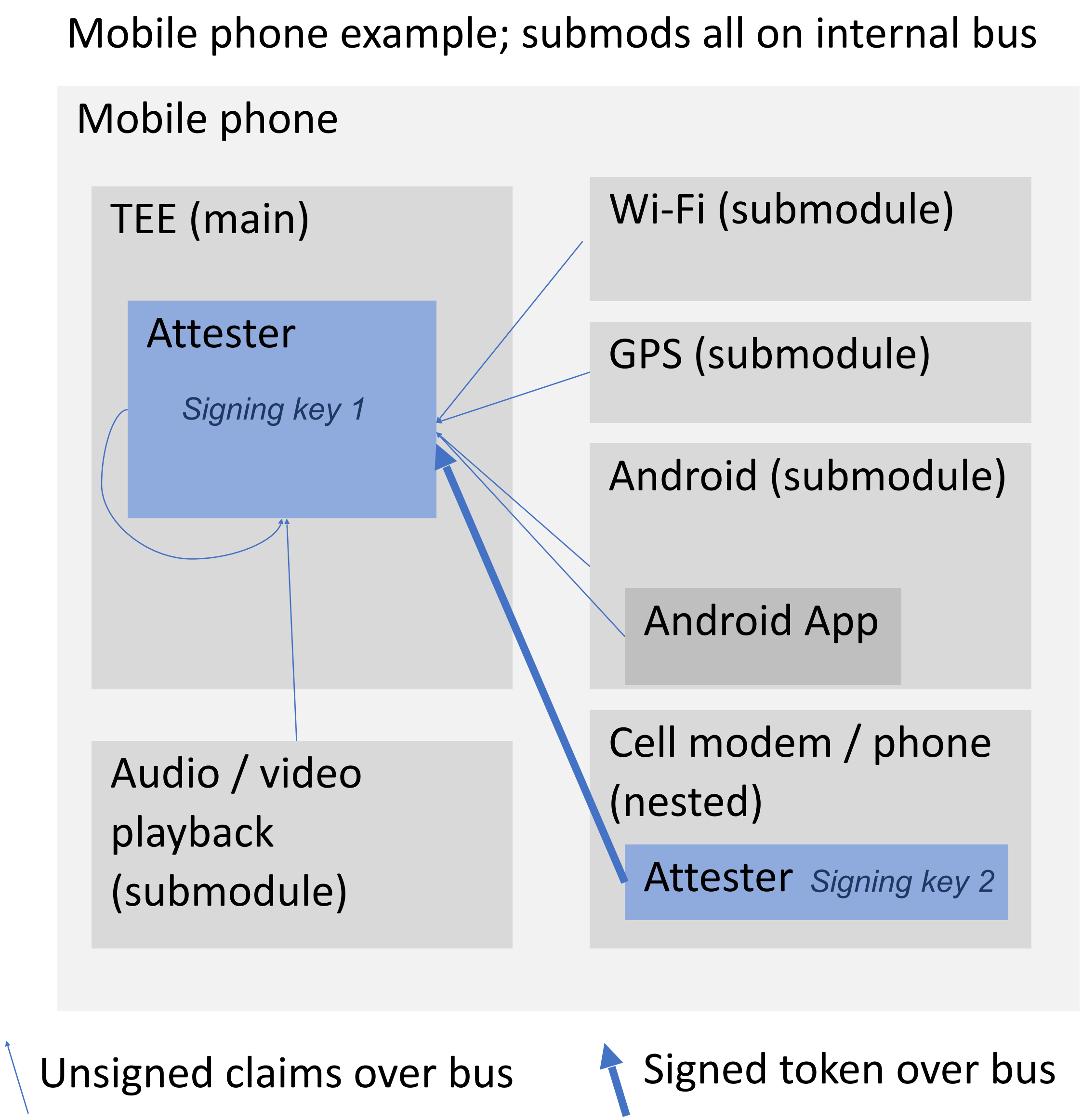
enabled	Debug is currently disabled, but may have been previously enabled
disabled	Debug has not been enabled in this boot cycle, but may have been enabled in previous boot cycles
disabled-since-boot	Debug is currently disables and has been so since boot
disabled-permanently	Only OEM identified by the OEMID can enable debug
disabled-fully-and-permanently	No debug facility can be enabled

Boot Seed

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

Submodules



- 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…
- 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)

Abbreviated Submods Example

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

   / submods / 20:
      {
         / 1st submod, an Android Application / “App Foo”: {
           / nonce /                  7:h'948f8860d13a463e8e',
           / seclevel /      11:1, / unrestricted /
           / 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 /
            / custom - release / -80000:'8.0.0',
            / custom - version / -80001:'4.9.51+'
         }
      }
}
```

Manifests claim

- This claim contains SW manifests that originated *outside the device,* typically from the SW provider or distributor
- Carries manifests defined by other than EAT — EAT doesn't define a manifest format
    - CoSWID
    - SUIT manifest
- The manifest may or may not be signed by its originator depending on the format and the originator
- The attestation signature will cover the entire manifest

SW Evidence Claims

- This claim contains SW measurements that originated *on the device*
- Carries measurements sets defined by other than EAT — EAT doesn't define a measurements format
  - CoSWID
  - CoRIM/CoMID
- Typically signed as part of attestation signing, but additional signing that is part of some measurements system is not excluded

SW Measurement Results Claims

- This is a proposed claim, not in any draft (just a PR in GitHub)
- Primarily aimed at conveying the results of a comparison of a measurement to reference values to the relying party
- Also can convey results of a completed on-device measurements system to the verifier
  - The device must have reference values for this to work
  - An example of a system like this is Samsung TIMA

Geographic Location -- WGS84 Coordinate System

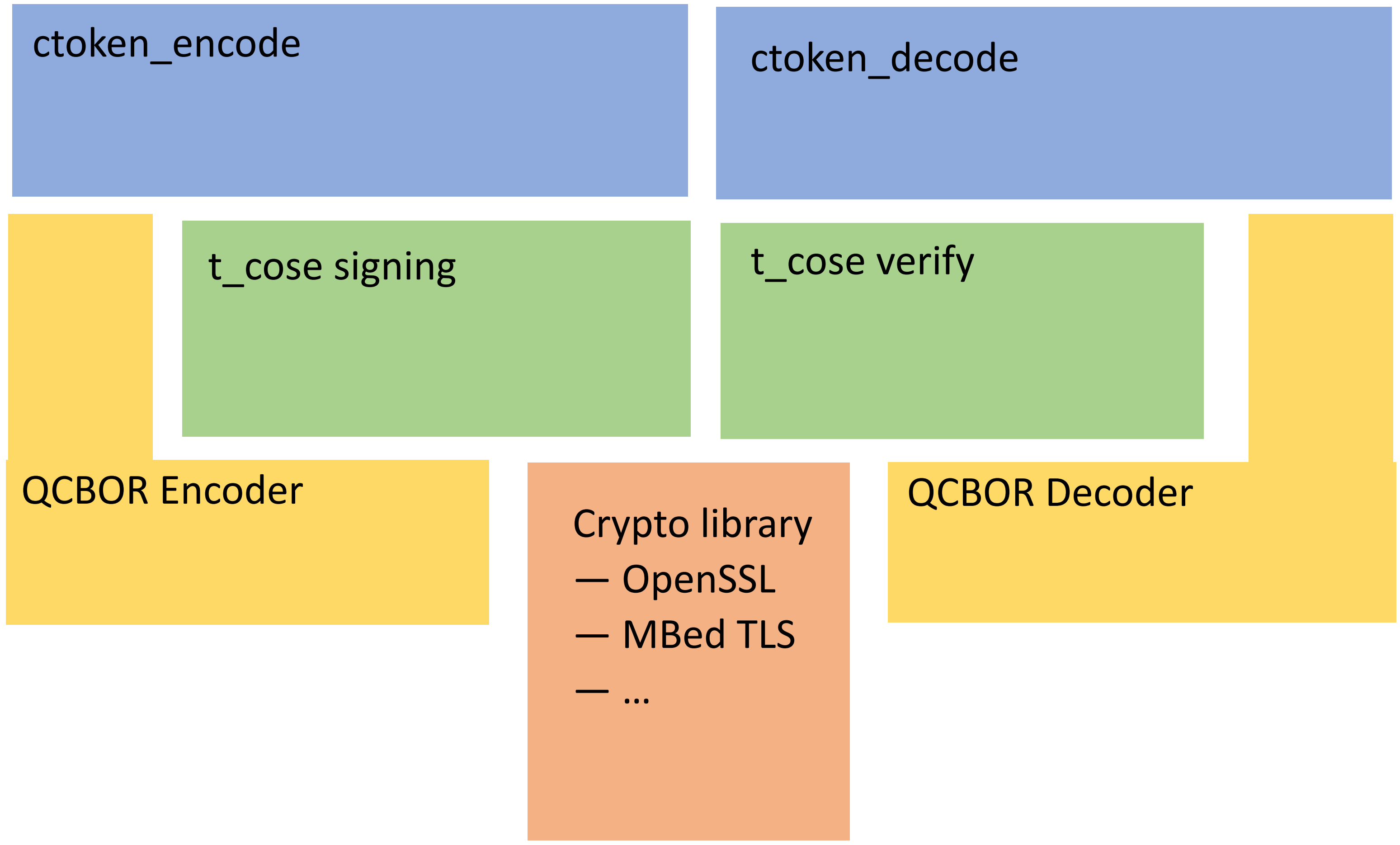
Latitude	
Longitude	
Altitude	
Accuracy	Accuracy of latitude and longitude in meters
Altitude accuracy	Accuracy of altitude in meters
Heading	0 to 360
Speed	Meters/second

Profiles

- A Profile is document that narrows the definition of an EAT token for a use case
  - Automotive, payment system, mobile phone,…
- Usually necessary to achieve interoperability
- May specify
  - Encoding format (e.g., only CBOR)
  - Cryptographic algorithms (COSE is open-ended)
  - Mandatory claims
  - Prohibited claims
- A specific claim identifies the profile used by the EAT

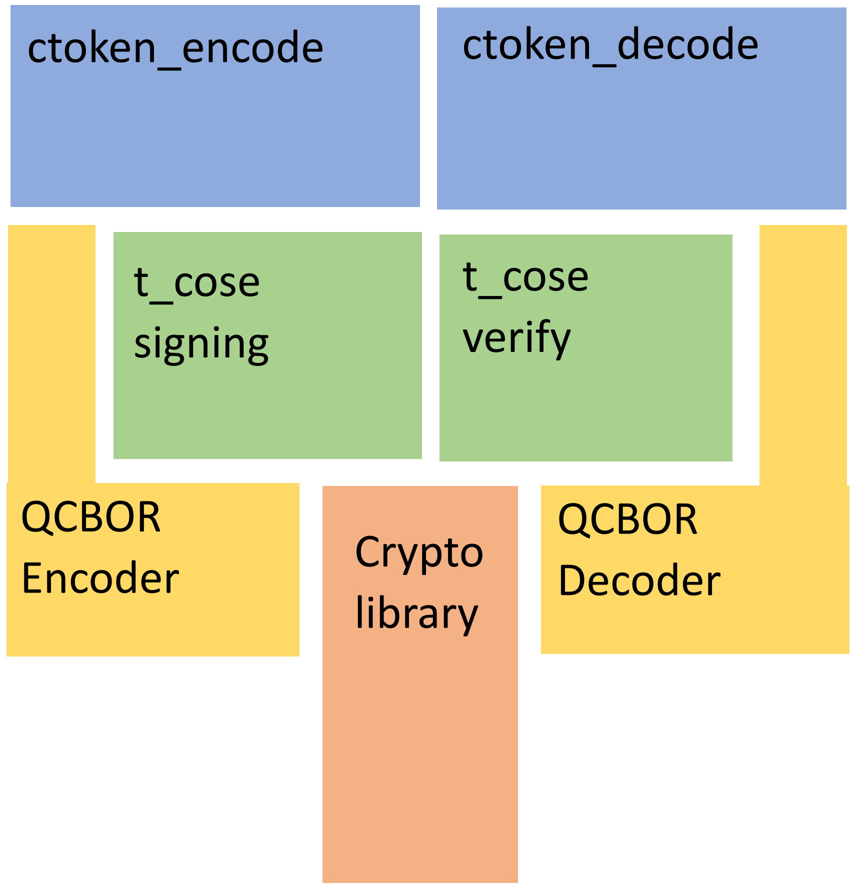
EAT
Open source SW stack

Cake Diagram for ctoken / t_cose / QCBOR



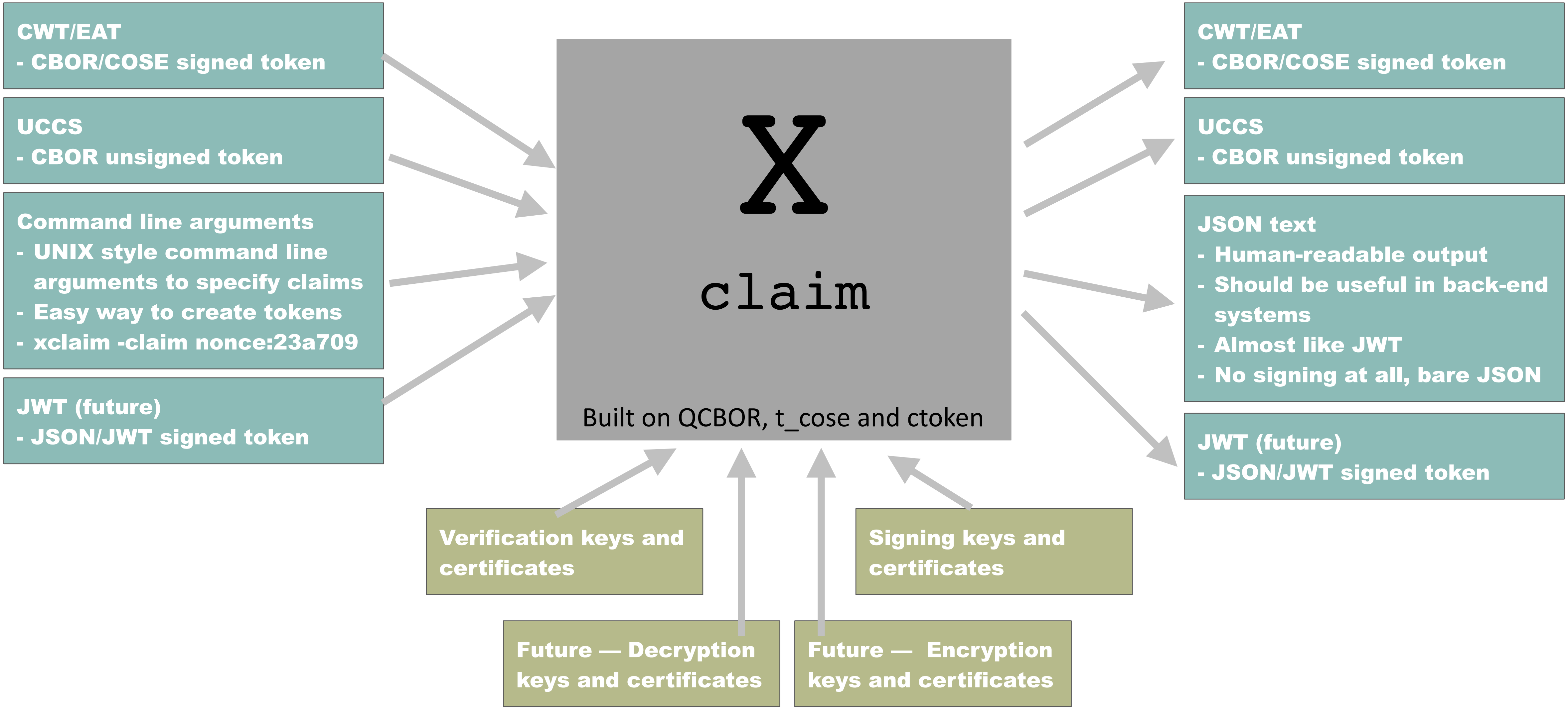
- ctoken
  - Implements EAT, CWT and UCCS
  - Standard claims
  - Easy to add new claims
- t_cose
  - COSE signing
  - Crypto adaptor layer
    - OpenSSL and MBedT TLS supported now
    - Others can be added
- QCBOR
  - Commercial quality full CBOR implementation

Code size and status

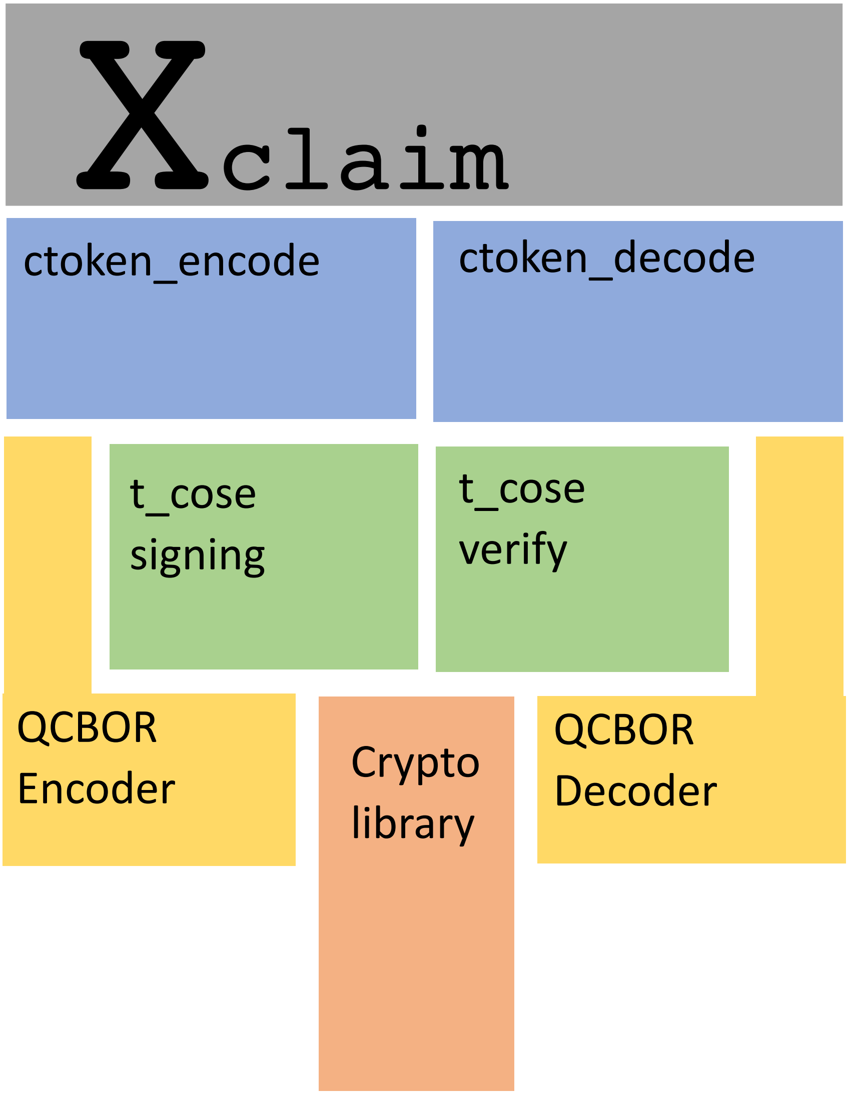


- Supported as commercial quality
  - Thorough tests
  - Backwards compatibility is maintained
  - Documentation
- Suitable for embedded
  - Small code size
  - Small memory requirements
  - No use of malloc
  - Encode / decode split neatly for use cases that need only one
- ctoken is not as quite as mature as t_cose and QCBOR

xclaim — command line tool for converting claim sets



GitHub Projects

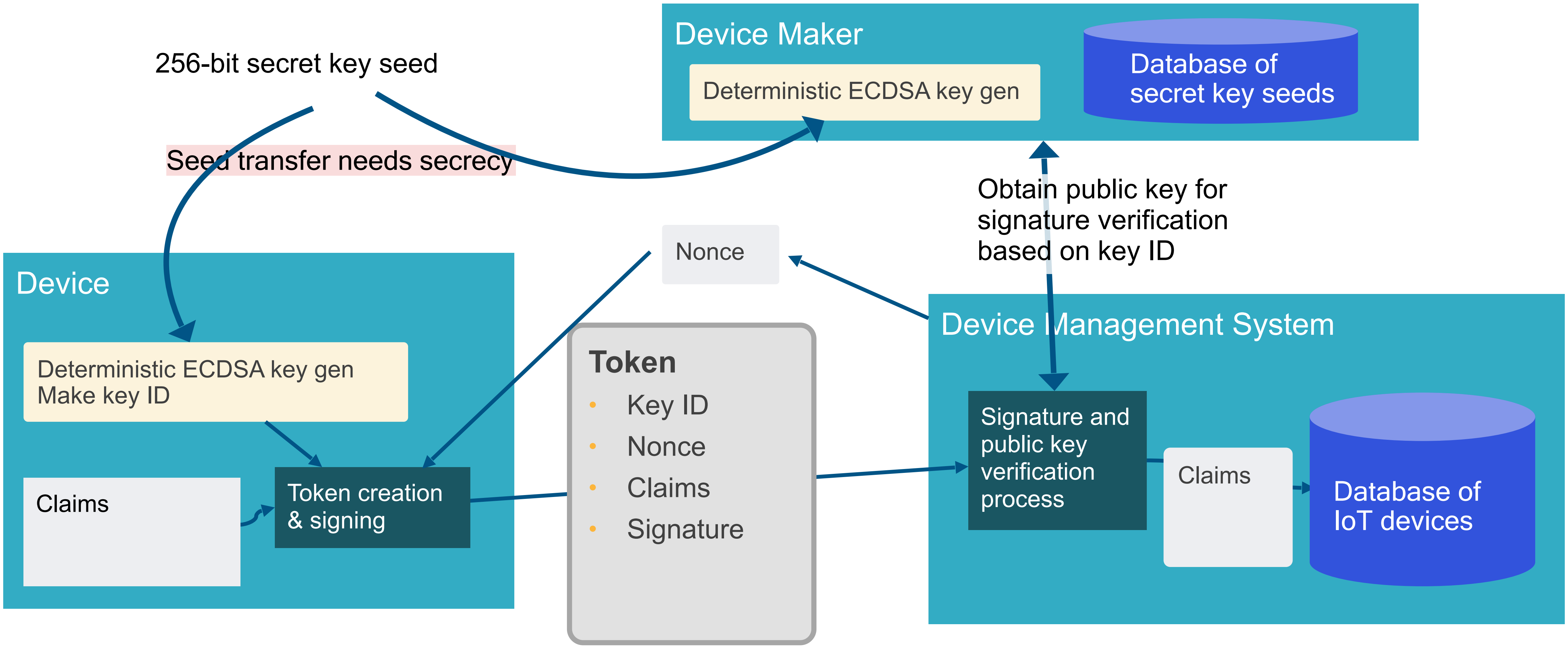


- https://github.com/laurencelundblade/QCBOR
- https://github.com/laurencelundblade/t_cose
- https://github.com/laurencelundblade/ctoken

- https://github.com/laurencelundblade/xclaim
  - `xclaim` is still in-progress
  - Basics are working
  - Not all claims are working
  - More crypto algorithms and key types can be added
  - …

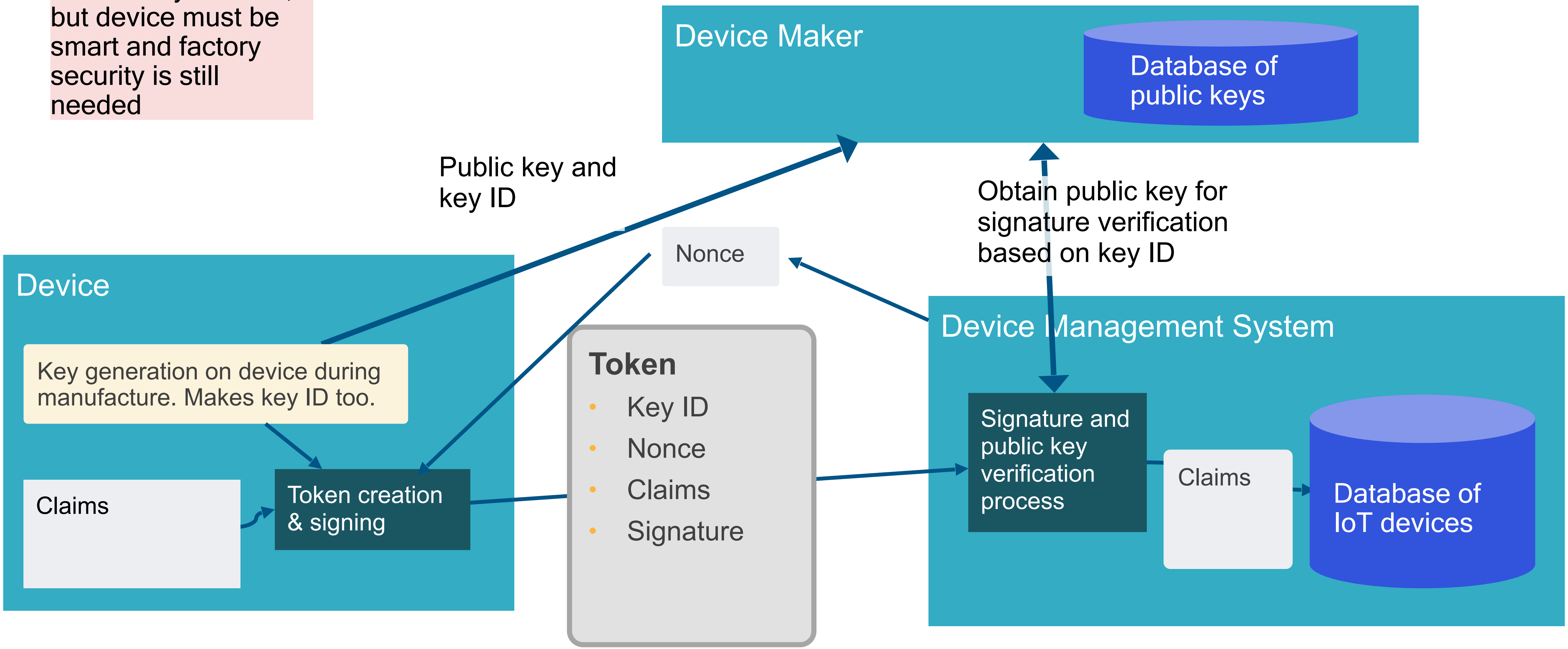
Examples of Signing Key Setup

ECDSA key setup based on 256-bit secret seed



ECDSA key setup generating key on device

No secrecy needed, but device must be smart and factory security is still needed



ECDSA key setup generating key outside of device