RATS Virtual Interim Feb 2020

EAT Claims Discussion

Laurence Lundblade

UEID Size Discussion

UEID sizing is not the same as for IP addresses

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

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

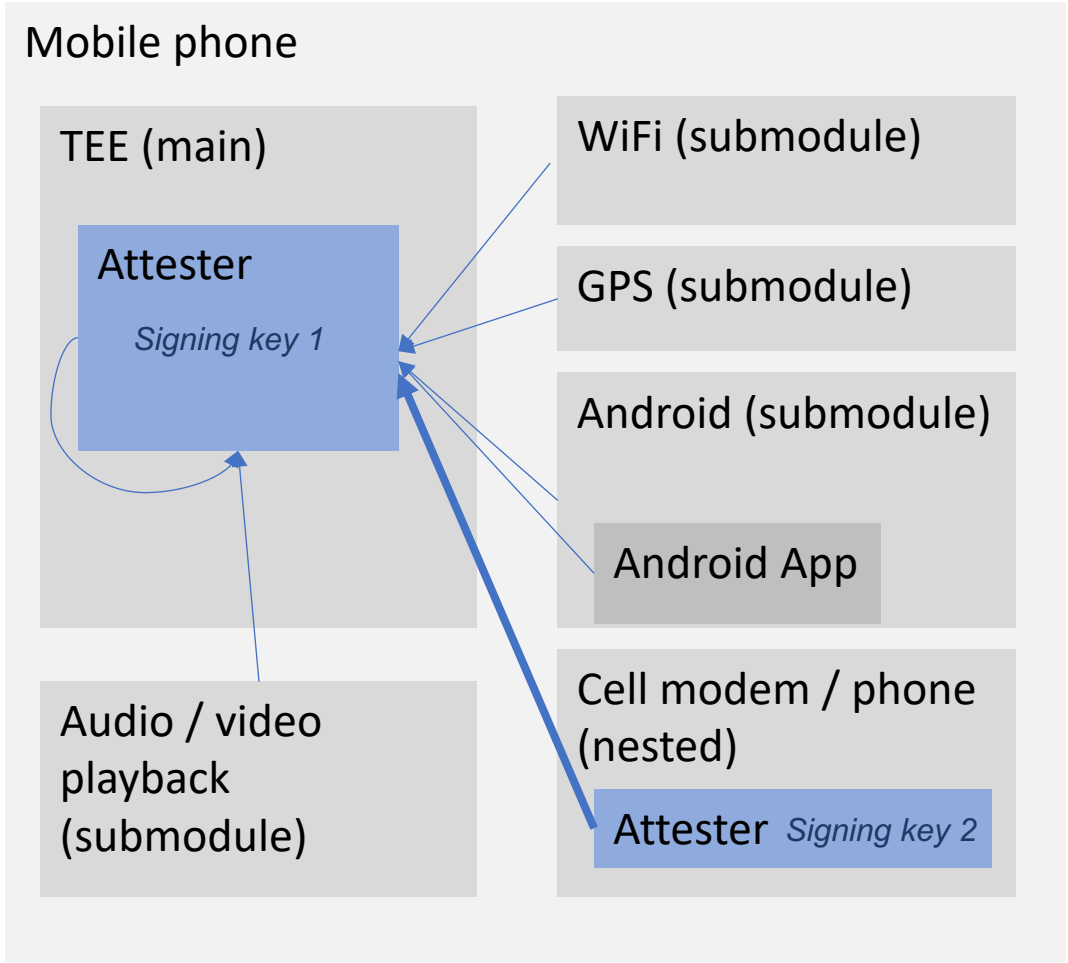
Database size	128 bits	192 bits	256 bits
trillion	2 * 10^-15	8 * 10^-35	5 * 10^-55
quadrillion	2 * 10^-09	8 * 10^-29	5 * 10^-49
100 quadrillion	2 * 10^-05	8 * 10^-25	5 * 10^-45

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

PR – New `submods` structure

Mobile phone example; submods all on internal bus



Unsigned claims over bus

Signed token over 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

Submods Example

```
/ submods / 20: {
        “wifi” : {
                / attachment_type / 16: 3 / pcb_internal /
                / nonce /  7:h’87f0e6…’
                / oemid /   12:h’653a… ‘ / The OUI of the WiFi maker /
        }

        “audio” : {
                / attachment_type / 16: 4 / chip_internal /
                / nonce /  7:h’87f0e6…’
                / oemid /   12:h’6c4573a… ‘ / The OUI of the audio maker /
        }
        “modem” : / A full nested and signed EAT (not shown) /
```

Alternate Submods Example

```
/ submods / 20: [
        [ / Array of three things: attachment type, name and claims /
                3 / attachment_type  pcb_internal /
                “wifi” / Name of subsystem /
                {
                        / nonce /  7:h’87f0e6…’
                        / oemid /   12:h’653a… ‘ / The OUI of the WiFi maker /
                }
        ],

        [ / Array of three things: attachment type, name and claims /
                4 / attachment_type  chip_internal /
                “audio” / Name of subsystem /
                {
                        / nonce /  7:h’87f0e6…’
                        / oemid /   12:h’6c4573a… ‘ / The OUI of the audio maker /
                }
        ],
        [ / Array of three things: attachment type, name and nested EAT /
                4 / attachment_type  chip_internal /
                “modem” / Name of subsystem /
                <> / full nested EAT, not shown /
        ]
]
```

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_reported: 0,
    not_disabled: 1,
    disabled: 2,                 May have been enabled earlier
    disabled_since_boot: 3,
    permanent_disable: 4,        Only the manufacturer can enable
    full_permanent_disable: 5    Not even the manufacturer can enable
)
```

Discussion on debug states

```
debug_disable_level = (
    not_reported: 0,
    not_disabled: 1,
    disabled: 2,                 May have been enabled earlier
    disabled_since_boot: 3,
    permanent_disable: 4,        Only the manufacturer can enable
    full_permanent_disable: 5    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.

Claims Characteristics PR, slide 1

General advice on claim design; may relates more to IANA registry

- 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
  - Include 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