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

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

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

<table>
<thead>
<tr>
<th>People</th>
<th>Devices/person</th>
<th>subsystems / device</th>
<th>Database portion of population</th>
<th>Resulting database size</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 billion</td>
<td>100</td>
<td>10</td>
<td>.1</td>
<td>trillion</td>
</tr>
<tr>
<td>10 billion</td>
<td>100,000</td>
<td>10</td>
<td>.1</td>
<td>quadrillion</td>
</tr>
<tr>
<td>100 billion</td>
<td>1,000,000</td>
<td>10</td>
<td>.1</td>
<td>100 quadrillion</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Database size</th>
<th>Database size</th>
<th>128 bits</th>
<th>192 bits</th>
<th>256 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>trillion</td>
<td>2 * 10^-15</td>
<td>8 * 10^-35</td>
<td>5 * 10^-55</td>
<td></td>
</tr>
<tr>
<td>quadrillion</td>
<td>2 * 10^-09</td>
<td>8 * 10^-29</td>
<td>5 * 10^-49</td>
<td></td>
</tr>
<tr>
<td>100 quadrillion</td>
<td>2 * 10^-05</td>
<td>8 * 10^-25</td>
<td>5 * 10^-45</td>
<td></td>
</tr>
</tbody>
</table>

### Time to collision assuming 10% of database changes per year

<table>
<thead>
<tr>
<th>Database size</th>
<th>Database size</th>
<th>128 bits</th>
<th>192 bits</th>
<th>256 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>trillion</td>
<td>60,000 years</td>
<td>10^24 years</td>
<td>10^44 years</td>
<td></td>
</tr>
<tr>
<td>quadrillion</td>
<td>8 seconds</td>
<td>10^14 years</td>
<td>10^34 years</td>
<td></td>
</tr>
<tr>
<td>100 quadrillion</td>
<td>8 microseconds</td>
<td>10^11 years</td>
<td>10^31 years</td>
<td></td>
</tr>
</tbody>
</table>
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 signing 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…' / The OUI of the WiFi maker /
            / 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…' / The OUI of the audio maker /
            / 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:

```python
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

```python
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,  \(\text{May have been enabled earlier}\)
    disabled_since_boot: 3,
    permanent_disable: 4,  \(\text{Only the manufacturer can enable}\)
    full_permanent_disable: 5 \(\text{Not even the manufacturer can enable}\)
}

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...)
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