Discovery for BRSKI
draft-eckert-anima-brski-discovery-01
ANIMA WG IETF118

Toerless Eckert, Futurewei USA (tte@cs.fau.de)
David von Oheimb (david.von.oheimb@siemens.com)
Esko Dijk (esko.dijk@iotconsultancy.nl)
11/07/2023
Coalesce “advanced discovery” for BRSKI drafts

- Ongoing drafts of WG
  - Discovery required by / beneficial for
    brski-ae, brski-cloud, constrained-brski (voucher, proxy), prm

- RFC8995 has no framework to allow for discovery of BRSKI variations
  - Did not predict non-interoperable variations between components.

- Attempts for defining separate extensions in each BRSKI draft is piece meal.
  - Duplicate text, unclear where to write what, ...

- This draft proposes to define discovery via common draft
  - Moves text from existing draft, defines common framework / solution
- Similar to splitting up prior BRSKI work, e.g.: brski-ae / jws-voucher
  - Adopt directly requested.
Example: Pledge discovering registrar/proxy

- Example brski-ae: Can not reuse same service name alone
  - What happens if a CMP-only registrar would announce itself via “brski-registrar”
  - An RFC8995 only (EST) pledge would discover it and fail
  - Sure, pledge would hopefully continue to try other discovered (EST) registrars
  - But not good design to have planned failures (unless problem is not easier to solve)
  - And there are simpler solutions

- New service-name - “brski-registrar-lwcmpp” to discover CMP capable registrar
  - To get brski-ae to RFC before we can finish a general solution

- Would need to introduce new service names for every COMBINATION of new BRSKI variations.
  - Does not scale
  - Would over time raise eyebrows with IANA service-name registry

- BRSKI proxies could not automatically determine what type of BRSKI variations exist
  - Even though they could perfectly act as proxies for all of them
Doc scope

• Variations
  • Pledges/Proxies/Registrar-Proxies discovering Registrars/Proxies
  • Extensible to current/future discovery cases
    • Registrar-Agents discovering Pledges
    • Other ? Registrars discovering MASA ? ...

• Make Proxies automatically work for variations

• Define once: Variation and encoding across different discovery protocols
• Extend via new BRSKI discovery registry tables

• Discovering pledges details/requirements - extended/refined from BRSKI-PRM
Known type of variations today
Pledge <-(proxy)->Registrar BRSKI variations

- Initiator/responder “mode”:
  - rrm  – RFC8995: “Registrar Responder Mode”
  - prm  – “Pledge Responder Mode” (additional requirements against registrar)

- Voucher format “vformat”:
  - cms  – RFC8995 CMS signed JSON voucher
  - cose – CBOR with COSE signature voucher (constrained voucher)
  - jose – JOSE signed JSON, preferred by PRM solutions

- Enrollment protocol “enroll”:
  - est  – RFC7030 as defined for RFC8995
  - cmp  – Primary EST alternative defined by BRSKI-AE (AE explains how to define more)
  - scep – example alternative, RFC8894. Not yet asked for with BRSKI.
  - ... – easily more (some mentioned in brski-ae)

Ideally should be able to make discovery work for every combination
And not write new drafts for discovery for any new options / combinations!

Networks need to support deployment of “ships-in-the-night”
E.g.: CMP registrars/pledge, PRM registrars/agents/pledges, RFC8995 registrar/pledges.

Do not force any implementation to ALSO implement variations that they do not care about.
Discovery mechanism variations

• **GRASP**
  • Original ANIMA/ANI discovery method

• **CORE-LF / Directory**
  • Redefined HTTP “Web Linking” (RFC5988) as a CORE discovery mechanism (RFC6690).
  • Also extended now with directory options

• **DNS**
  • DNS-SD via mDNS: Was multi-hop, nowadays should only be link-local. Some networks define multi-hop
  • DNS-SD with Unicast DNS and SRP (dynamic unicast registration of DNS/DNS-SD information)

Can not predict what methods individual deployments prefer

• Example: CORE-LF designed for “IoT” networks. Assuming “constrained” BRSKI would only need CORE-LF
• But even IoT network deployments may prefer DNS-SD
  • DNS always required in network, when there are also human users.
• Constrained protocols likely also to be used in unconstrained networks
  • Implement once in constrained and unconstrained devices. Deploy everywhere
• Also want to make sure GRASP can get used, even if not preferred today.
Proxies automatically supporting variations

- BRISKI Proxies only forwards TCP/UDP connection packets
  - Do not care about what messages/exchanges happen inside.
  - But only those messages make pledge/registrar compatible or incompatible

- Proxy needs to be able to discovery variations supported by registrar
  - Announce equivalent proxy-service variations
  - Connect pledge connections for a specific proxy-variation to the right registrar

- Details specified in draft.
  - Can be non-obvious: Registrar A support variation 1,2, Registrar B supports variation 2,3. Proxy may create 3 sockets, one for each variation, and when it receives connection for a socket pick Registrar A and/or B – depending on socket.
Registry tables (1)

BRSKI Variation contexts

<table>
<thead>
<tr>
<th>Context</th>
<th>Applicable Service Name(s)</th>
<th>Discovery Variation Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRSKI</td>
<td>mode vformat</td>
<td>GRASP</td>
</tr>
<tr>
<td></td>
<td>enroll with IPPROTO_TCP</td>
<td></td>
</tr>
<tr>
<td>cBRSKI</td>
<td>mode vformat</td>
<td>GRASP</td>
</tr>
<tr>
<td></td>
<td>enroll / with IPPROTO_UDP</td>
<td></td>
</tr>
<tr>
<td>BRSKI-PLEDGE</td>
<td>vformat</td>
<td>DNS-SD</td>
</tr>
<tr>
<td></td>
<td>enroll</td>
<td>CORE-LF</td>
</tr>
</tbody>
</table>
Registry tables (2)

BRSKI Variation Type Choices

- Defines all variation type choices (rrm, prm, cms, cose, ...)
  - Structured by variation-type (mode, vformat, enroll)
  - Structured by context
    - Different contexts may allow different subsets and may have different defaults

<table>
<thead>
<tr>
<th>Context</th>
<th>Variation Type</th>
<th>Variation Type Choice</th>
<th>Reference</th>
<th>Flags</th>
<th>Note(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRSKI, cBRSKI</td>
<td>mode</td>
<td>rrm</td>
<td>[RFC8995]</td>
<td>Dflt</td>
<td>Registrar Responder Mode the mode specified in [RFC8995]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ThisRFC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRSKI</td>
<td>vformat</td>
<td>cms</td>
<td>[RFC8368]</td>
<td>Dflt</td>
<td>CMS-signed JSON Voucher</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ThisRFC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>cose</td>
<td>ThisRFC</td>
<td></td>
<td>CBOR with COSE signature</td>
</tr>
</tbody>
</table>
Registry tables (3)

BRSKI Variations

A flat string called "variation" for every applicable combination of variation type choices

Well defined creation method – but safer to still create an explicit table – so coders can not misread the spec.

<table>
<thead>
<tr>
<th>Context</th>
<th>Spec / Applicability</th>
<th>Variation String</th>
<th>Variation Choices</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRSKI</td>
<td>[RFC8995]</td>
<td>&quot;EST-TLS&quot;</td>
<td>est</td>
</tr>
<tr>
<td>Note 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;EST-TLS&quot;</td>
<td>est</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;EST-TLS&quot;</td>
<td>est</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;EST-TLS&quot;</td>
<td>est</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;EST-TLS&quot;</td>
<td>est</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>possible variation of [RFC8995] with</td>
<td>jose</td>
<td>rrm jose</td>
<td></td>
</tr>
<tr>
<td></td>
<td>basic-anima-brski-ae</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>possible variation of [RFC8995] with</td>
<td>jose-cmp</td>
<td>rrm jose</td>
<td></td>
</tr>
<tr>
<td></td>
<td>basic-anima-brski-ae</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>possible variation of [RFC8995] with</td>
<td>jose-cmp</td>
<td>rrm jose</td>
<td></td>
</tr>
<tr>
<td></td>
<td>basic-anima-brski-ae</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>possible variation of [RFC8995] with</td>
<td>jose-cmp</td>
<td>rrm jose</td>
<td></td>
</tr>
<tr>
<td></td>
<td>basic-anima-brski-ae</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
GRASP examples

All from proxy – registrar announcements would use “AN_join_registrar”

```
[M_FLOOD, 12340815, h'fe800000000000000000000000000001', 180000,
 ["AN_Proxy", 4, 1, "", [O_IPV6_LOCATOR,
  h'fe800000000000000000000000000001', IPPROTO_TCP, 4443],
 ["AN_Proxy", 4, 1, "jose-cmp", [O_IPV6_LOCATOR,
  h'fe800000000000000000000000000001', IPPROTO_TCP, 4443],
 ["AN_Proxy", 4, 1, "", [O_IPV6_LOCATOR,
  h'fe800000000000000000000000000001', IPPROTO_UDP, 4684]]
 ]
```

```
[M_FLOOD, 42310815, h'fe800000000000000000000000000001', 180000,
 ["AN_Proxy", 4, 1, "prm", [O_IPV6_LOCATOR,
  h'fe800000000000000000000000000001', IPPROTO_TCP, 44000]]
]
```

RFC8995

jws+cmp

cbrski

prm from separate app, separate socket
DNS-SD example

_brski-registrar._tcp.local  IN PTR  0200:0000:7400._brski-registrar._tcp.local
0200:0000:7400.
_brski-registrar._tcp.local  IN SRV  1 2 4443  0200:0000:7400.local
0200:0000:7400.
_brski-registrar._tcp.local  IN TXT  "dflt" "jose-cmp"

_brski-registrar._udp.local  IN PTR  0200:0000:7400._brski-registrar._udp.local
0200:0000:7400.
_brski-registrar._udp.local. IN SRV  1 2 4684  0200:0000:7400.local
0200:0000:7400.
_brski-registrar._udp.local  IN TXT  "dflt"

0200:0000:7400.local  IN AAAA  fe80:0000:0000:0000:0000:0000:0000:0001

_brski-registrar._udp.local  IN PTR  0200:0000:7400-prm._brski-registrar._udp.local
0200:0000:7400-prm.
_brski-registrar._udp.local. IN SRV  1 2 44000.  0200:0000:7400-prm.local
0200:0000:7400-prm.
_brski-registrar._udp.local  IN TXT  "prm"

0200:0000:7400-prm.local  IN AAAA  fe80:0000:0000:0000:0000:0000:0000:0001
DNS vs. GRASP

- Use same variations strings
  - DNS-SD does not allow empty string, use “dflt” instead
- DNS-SD works via “Service Instance Names”
  - For human readability.
  - Also used to express different sockets with different services on same host.
  - Not in currently defined GRASP objectives (not needed)
- DNS-SD also has “target names”
  - Allows each service instance to be bound to multiple addresses (IPv4/IPv6)
  - If Service does have multiple addresses, then GRASP would need multiple service lines for these

- Just need to understand the peculiarities
  - Draft tries to explain them based on different examples and requirement details.
Pledge discovery

• For BRSKI-PRM - refined/enhanced text vs. PRM draft
  • Eg: expand from mDNS to unicast DNS with Service Registration
  • Useful also in absence of PRM – discover pledge when RRM fails.

• Explicit DNS-SD details and examples

• Text tries to outline requirements against IDevID and sales-channel requirements so that PRM can be used
  • Sales information (“serial number”) needs to match IDevID information
  • IDevID information needs to be unique across product lines
  • ...

Ultimately we may want another RFC (maybe not even ANIMA) about IDevIDs... (Profile + extensions of 802.1AR)
Similar to IETF WebPKI X.509 RFCs... ?
Status

• Improved details in slides from work this week (-01 predates this)
  • BRISKI meeting monday, also finalized AE details...

• Would want to see moved to WG status
  • Outsourced from existing WG drafts

• Need to add / agree on encoding proposal for CORE-LF (CoAP discovery)
  • Esco provided suggestions
The End

• Asking WG chair to accept as WG draft.

• Questions?