RPKI Retrieval Delta Protocol - RRDP

Tim Bruijnzeels
Oleg Muravskiy
Bryan Weber
Rob Austein
David Mandelberg

IETF92 | Dallas | 23 March 2015

Update

- Draft accepted as working group document

- Some questions were raised in the working group
  - Design goals
  - HTTP vs other transport protocols
  - Quick overview

- Some questions were discussed between the authors
  - Implications of CAs sharing a publication point

- Implementation updates
  - Validation (almost) implemented (rcynic, RIPE NCC)
  - Publication server (open-source) planned (rpkid, RIPE NCC)

Design ideas 1/2

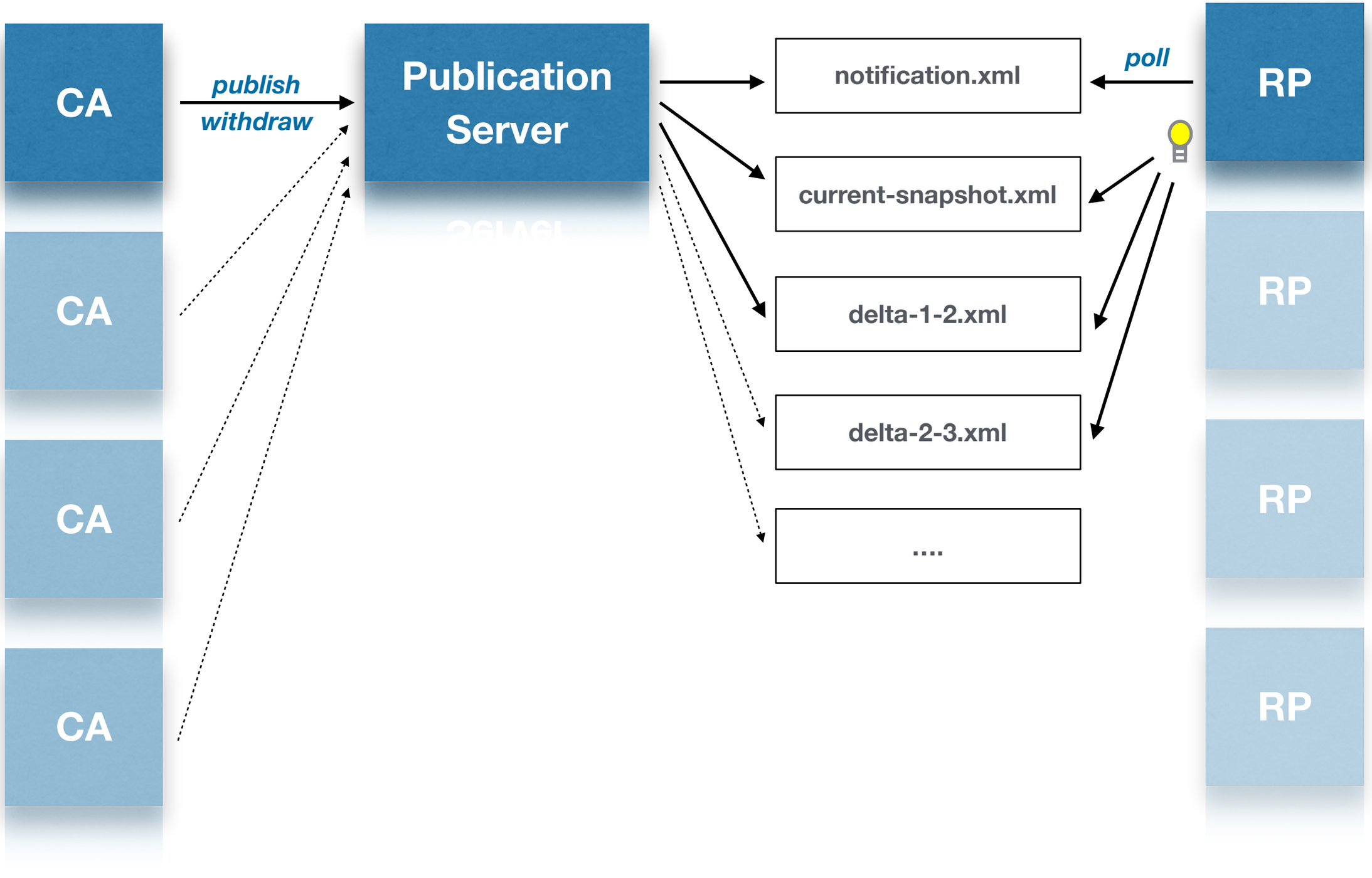
- Support publication servers
  - Many CAs share one publication server (compare recursive rsync)
  - Improve scaling: move expensive operations to the clients:
    - Finding deltas requires no interactive dialogue with server
  - HTTP(S) as primary transport
    - Industry standard, many server and library implementations
    - Support scaling by use of existing global http based CDNs
    - Support scaling by use of local http caching servers (squid, apache, etc)

Design ideas 2/2

- Support Relying Parties
  - Many mature native http(s) client libraries
  - Minimise number of fetches and amount of data

- Avoid *unnecessary* Complexity
  - Minimal protocol, clearly defined and versioned (no svn, git, etc)
  - No big attempts to secure the fetch protocol: trust object security (just like rsync)
  - Messages based on existing publication protocol draft

Quick overview



Quick overview - Publication

publication protocol

```
<messages>
      <publish uri="rsync://host/ca1/key.mft" hash="…">
            base64
      </publish>
      <publish uri="rsync://host/ca1/key.crl" hash="…">
            base64
      </publish>
      <publish uri="rsync://host/ca1/roa2.roa">
            base64
      </publish>
      <withdraw uri="rsync://host/ca1/roa1.roa" hash="…"/>

      ...

</messages>
```

Quick overview - RRDP snapshot / deltas

publication-protocol

Publication Server

new notify, listing current snapshot & deltas

```
<notify … session="…" version="…">
       <snapshot uri="…"  hash="…" />
       <delta version="…" uri="…" hash="…"/>
       <delta version="…" uri="…" hash="…"/>
       ...
</notify>
```

updated regularly at stable (https) URL

new delta -matching CA publication

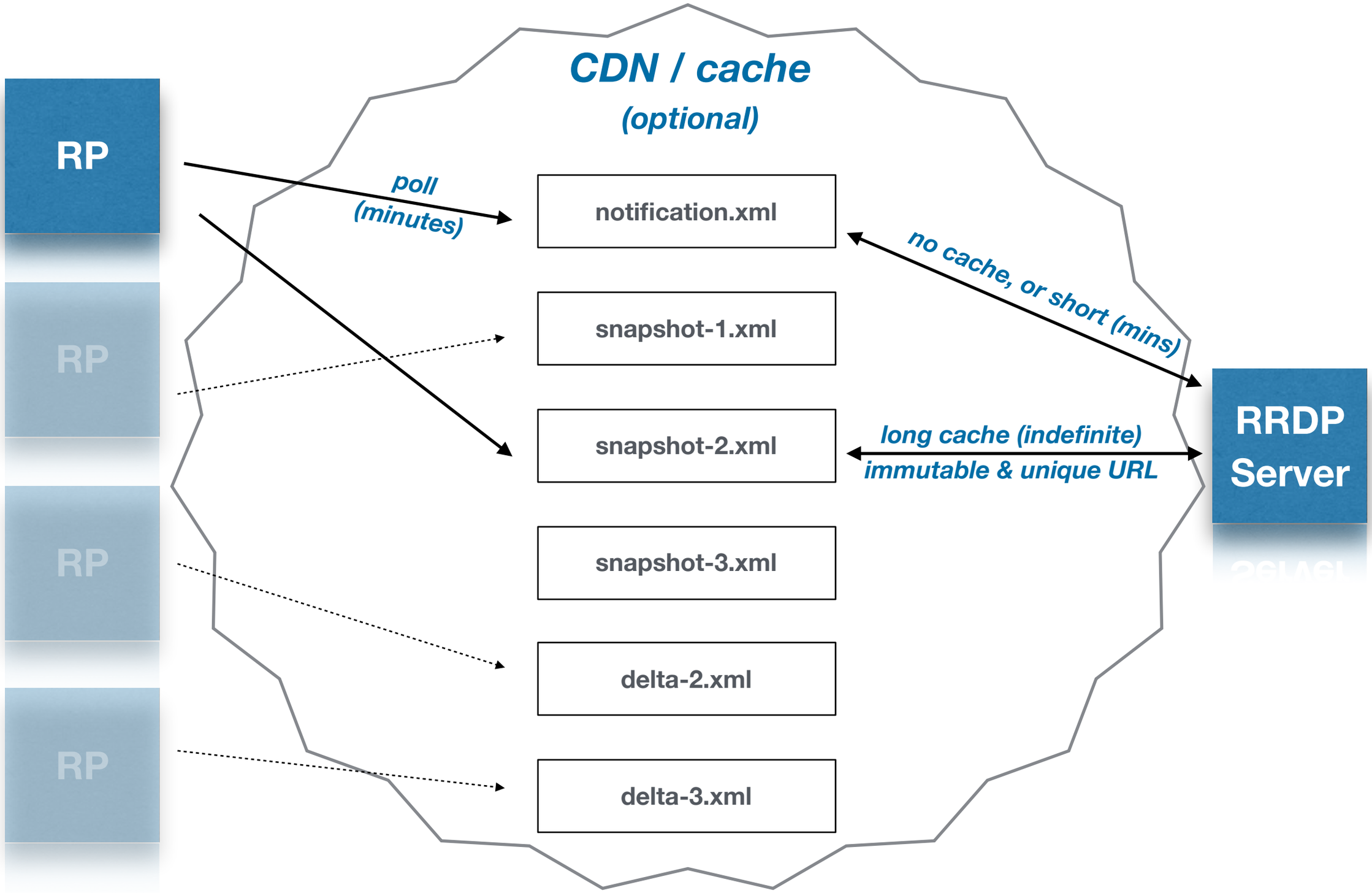
```
<delta … session="…" version="…">
       <publish uri="rsync://host/ca1/key.mft" hash="…">
              base64
       </publish>
       <publish uri="rsync://host/ca1/key.crl" hash="…">
              base64
       </publish>
       <publish uri="rsync://host/ca1/roa2.roa">
              base64
       </publish>
       <withdraw uri="rsync://host/ca1/roa1.roa" hash="…"/>

       ...

</delta>
```

new snapshot -all current objects, all CAs

```
<snapshot … session="…" version="…">
       <publish uri="rsync://host/ca1.cer">
              base64
       </publish>
       <publish uri="rsync://host/ca2.cer">
              base64
       </publish>
       <publish uri="rsync://host/ca1/roa2.roa">
              base64
       </publish>
       <publish uri="rsync://host/ca2/roa1.roa">
              base64
       </publish>
       ...
</snapshot>
```

Immutable data for a given session and version - published at unique (http) URLs to allow caching (CDN)

Quick overview - Validator polling / fetching



Implications of CAs sharing a publication point

- Additional SIA pointer with new OID

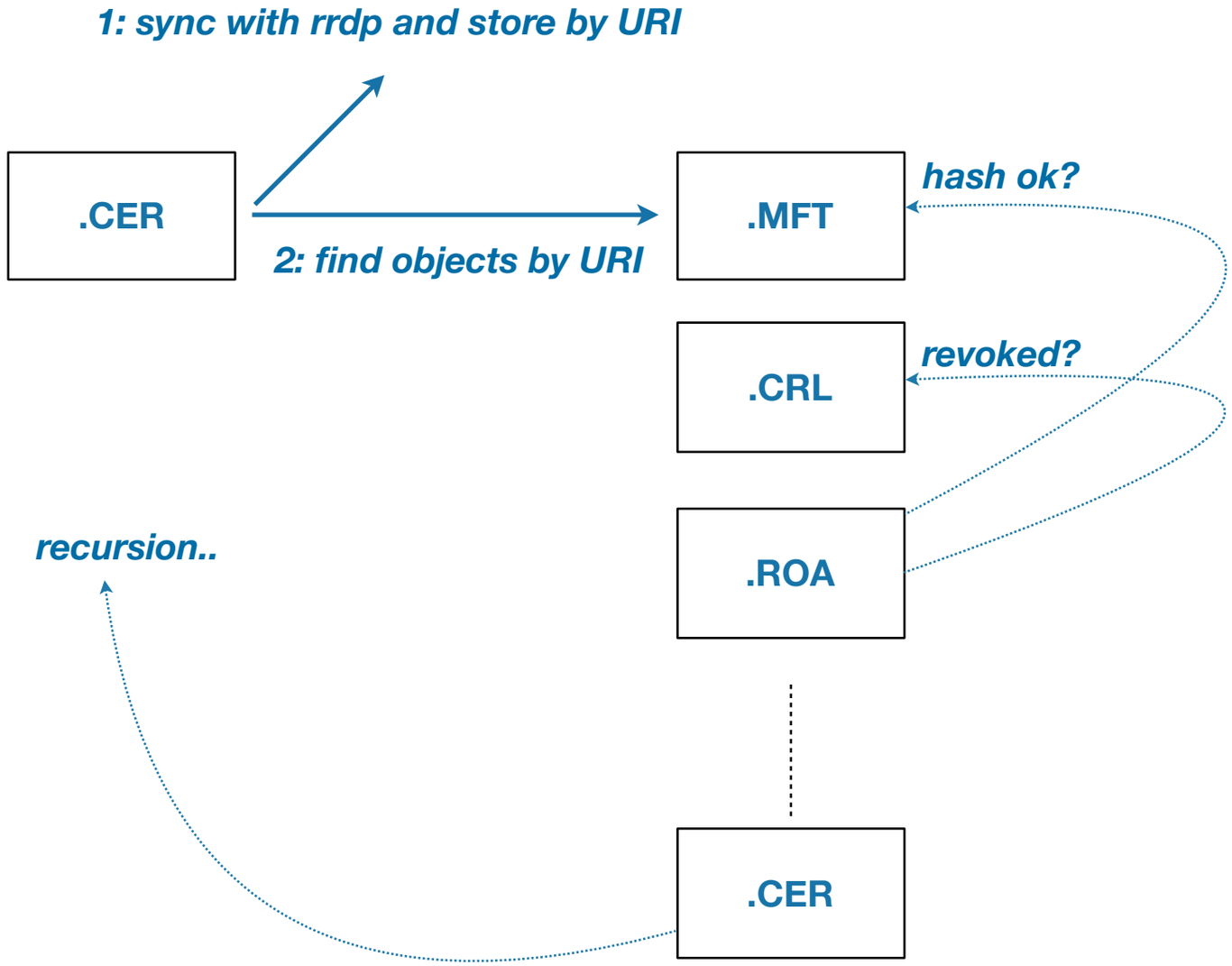
  - Points to notification.xml using http
    (will probably change to https)

  - Snapshots and deltas include ***other*** objects. How does an RP find the manifest and products signed by ***this*** CA certificate?

  - What about AIA and CRLDP? How does an RP find the parent CA certificate, and the CRL signed by the parent?

  - With rsync we expected the objects at exact the rsync URIs

Option 1: Use the URIs in the publish / withdraw elements



The quoted URI ***cannot*** be trusted

- Publication Server lies to RP
- or CA lies to sloppy Publication Server
- And pretend to have objects for another CA's URIs - *across publication servers*..

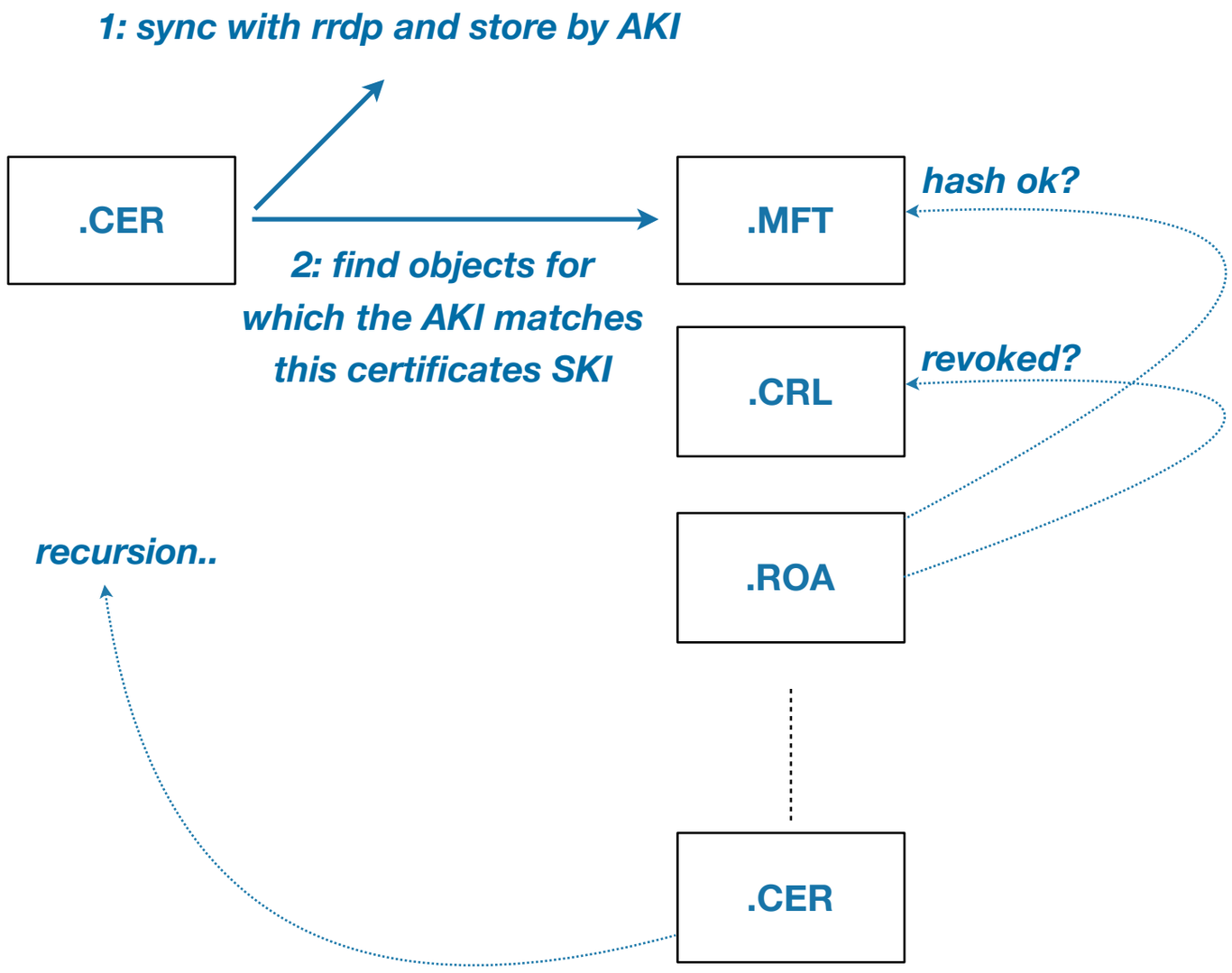
But.. you can't stop me from publishing my objects with my publication server with my URIs as well! I.e. you still cannot withhold an RP from obtaining my objects.

Solution?

Process **all** objects matching the URIs, but reject bogus objects (invalid signature, picture of kitten, whatever it may be).

One can lie about ***URIs***, but there is no proof of possession of the ***private key***!

Option 2: Discover objects by key identifiers



The quoted AKI ***cannot*** be trusted. Another AKI can be included, but.. the signature would be **invalid**.

How can an RP find the *current* manifest, CRL or certificate? (These objects are regularly replaced, but keep the same URI)

Solution?

- **Always** require valid signatures
- Use the **latest** CRL and MFT by **parent**
- Use the **latest** certificate for a given **SKI**

Use URIs for reporting and fetching, but not for validating

Advantage: multiple publication points..

Implementation Update - Validation

- Proof of concept code works (RIPE NCC and rcynic)

- Production grade code almost done for RIPE NCC validator
  - Validation and retrieval separated, handles rsync and rrdp
  - Using option 2.. relying on key identifiers for discovery
  - Still solving issues regarding forgetting about old objects (if not on mft and revoked, or not on mft and expired?)
  - Plans to write an informational document describing full validation and retrieval algorithm

Implementation Update - Publication

- Proof of concept code works (RIPE NCC and rpkid)

- RIPE NCC planning to work on open-source publication server
  - First version using publication protocol semantics
  - But not the CMS just yet (can easily be added later)
  - Supporting publishing to both RRDP and rsync
    (transparent to CA as long as it includes the right URIs)
  - Aim to have running code by IETF93

Questions? Comments?