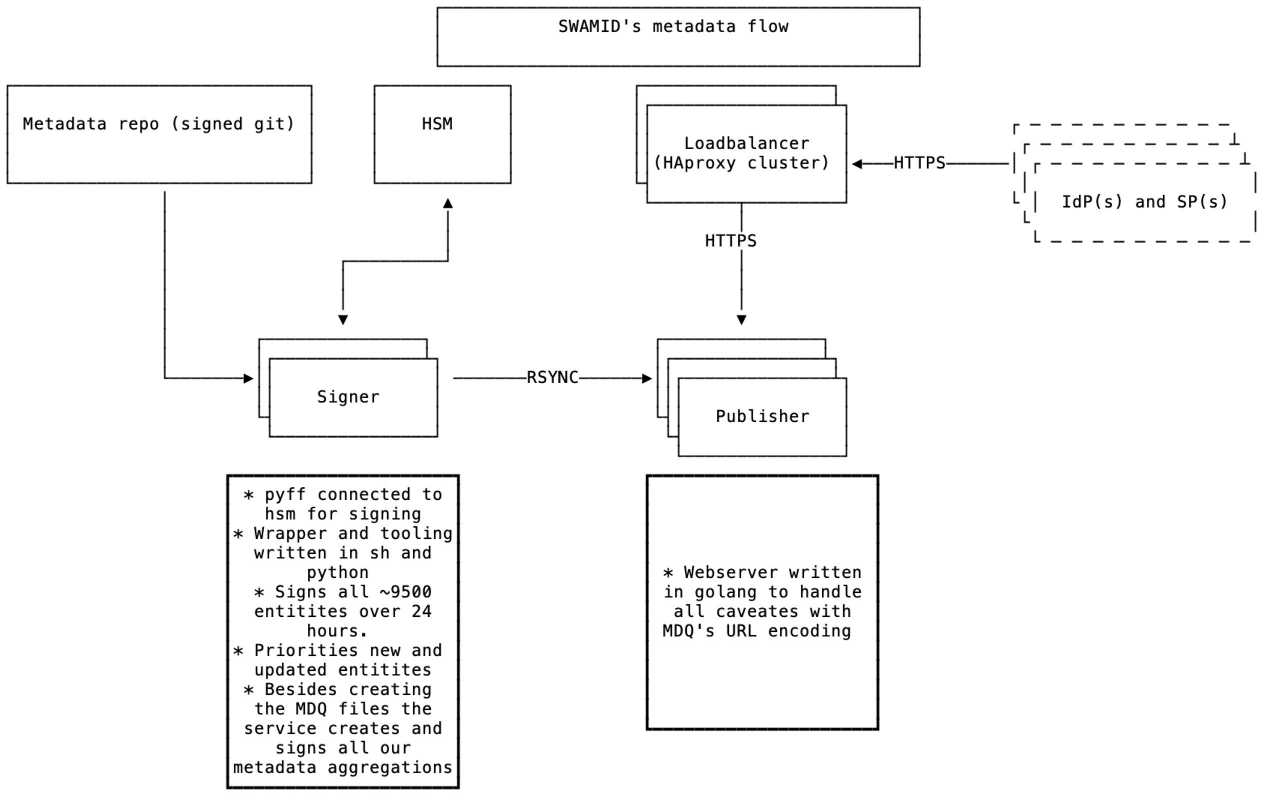
MDQ in SWAMID

Since 2016 SWAMIDs metadata has been signed with a key in a HSM. This has been fine since we previously only were creating a handful of metadata aggregation files (of different sizes). The problem with the aggregated feeds is that it takes time and memory(!) to load an in SWAMID's case 80 MB XML file in to the Identity Providers and Service Providers. Had a conversation recently with a new SP in our federation. They guessed that `shibd` crashed on start since it appeared to just hang - told them to take a deep breath and relax.

One solution for this memory and size problem isto start using the MDQ Protocol instead of big aggregated files. By using the MDQ protocol the SPs and IdPs don't need to load the whole federation. Instead they loads requested entities on the fly. So if both an IdP and an SP uses MDQ the SP will first fetch the IdPs metadata from the MDQ server and then redirect for login. The IdP will then fetch the SPs metadata from the MDQ server, prompt for authentication and then redirect the user back to the SP. Pretty easy flow but it requires the MDQ server to be fast and always available!

Our software of choice, pyFF, can act as a MDQ server and serve signed metadata files on request. But connecting `pyFF` to the HSM makes the signing too slow and would in a case of many requests in a short time create a heavy load on our HSM servers, which we would like to avoid. `pyFF` can be run in batch mode which will sign all entities and output them to disk (e.g for mirroring) but that would still require us to sign around 9500 entities each run which once again would put our HSMs at risk. Another factor why we chose the design we ended up with is that we would like to protect the machines (signers) connected to the HSM from the internet.



* pyff connected to hsm for signing
* Wrapper and tooling written in sh and python
* Signs all ~9500 entitites over 24 hours.
* Priorities new and updated entitites
* Besides creating the MDQ files the service creates and signs all our metadata aggregations

* Webserver written in golang to handle all caveates with MDQ's URL encoding

So what we came up with is tools and wrappers around `pyFF` which fetches SWAMIDs and eduGAINs metadata (a total of around 9500 entities), splits them up in parts, and signs all entities over the current day. That's around 400 per hour and we run it 4 times an hour via cron. This creates a reasonable load on the HSMs. We prioritises new, updated or removed entities which are usably published 15 minutes after we detect a change. The tooling is based on a `python` script we call mdqp which have som pre and post scripts written in `sh`.
The basic flow looks like:

- Metadata is fetched from git (a signed commit is verified) and eduGAIN
- `pyFF` reloads the metadata
- The given amount of entities (new, updated or removed prioritised) are fetched from `pyFF` via `mdqp`
- Our aggregated feeds are fetched from `pyFF`
- All signed entities and files are rsynced to the publishers (web servers)

There are two ways of getting an entity through the MDQ protocol:

- Getting them by entitiyId, e.g /entities/https://connect.eduid.se/sunet
- Getting them by the sha1sum of the entityid, e.g entities/{sha1}47918903a357c193bcd985a23c5958a8a43278c0

Both methods should be url encoded which makes things complicated. "Regular" webservers (e.g apache2 or nginx) decodes an incoming encoded url string before processing the request which would make it impossible for us to store the first alternative on disk (contains slashes). We also would like to support both alternative with the same file on disk so for that purpose we ended up writing ourself a very small and simple web server in golang which will serve our very niche set of requirements.

The web servers themselves are *protected* by a geo distributed cluster of HAProxy to even out the load and terminating TLS.

Happy MDQing.

See our wiki for more information about getting started with MDQ in SWAMID.

--
jocar

SWAMID Operations