DRP Security Model

We circulated a first draft of a technical security model in June as a catalyst for conversations, associated with the 0.5 version of the API specification and system rules. Many of you offered very valuable and comprehensive feedback. Thank you! We've attempted to capture and distill what you said.

With your feedback, and the progress of our own work on the technical, legal, and business fronts, we've put together and are proposing a roadmap both for the overall project, for conformance testing and interoperability testing, as well as for the security work.

However, because of how important security is to the project, we'll share here in a bit more detail how we hope to evolve the security model in the weeks and months ahead.

The Road Ahead

We envision security work unfolding in three phases—a testing phase (already underway) where we stick with the existing security model and prove out other aspects of the system (e.g. the ability to send requests end-to-end); an intermediate phase where we update the data model to capture all of the elements requiring signing and attestation in the signed block, but continue to use the existing signing primitive (JWTs); and a subsequent phase where we move to a signing primitive which is a drop-in replacement but also stronger cryptographically *and* much easier to use correctly (libsodium). In handy table form:

Phase	DRP version	Security Model Changes	Goal
Testing focus (underway)	v0.5	None (existing security model)	Prove out the system
Data model focus	v0.6	Update the data model incorporating security requirements	Ensure the data model and API provide the security guarantees we want
Cryptography focus	v0.7	Switch to libsodium	Improve security; improve ergonomics
Finalization focus	v0.8		Incorporate feedback from previous phases; finalize model

These phases correspond roughly from the transition from internal-only testing with dummy data in non-production systems (where we are now), to beginning to run actual customers' requests through actual production systems and actioning their requests.

They also correspond roughly to a transition from doing no consumer identity verification (because dummy data) to requiring email and offering phone number consumer identity verification with live consumer requests, and building the legal and operational machinery to accomplish and ensure that.

Our objectives are to:

1. Maximize the ease of integration of the DRP protocol into production systems
2. Provide very good security guarantees while doing so, and
3. Minimize or eliminate the need to rework the implementation once it's in production.

By the time the system is running production traffic, the security model should be mostly set, and we should be changing it to add new functionality or reduce operational burden rather than fixing the fundamentals.

This is part of a longer process of moving DRP towards a mature security model, but these first steps are essential to prove out whether we're on the right path or not.

Security Model for 1:1:1 Test (spec v.0.5)

For the forthcoming 1:1:1 test we plan to stick with the existing security model as implemented in e.g. OSIRAA, the open-source reference implementation of a DRP Authorized Agent.

Because only test data is running between non-production systems in the 1:1:1 test, non-technical identity verification is out of scope; we're just testing our ability to make requests end-to-end, from Authorized Agent all the way through to Covered Business, using the existing implementations and with the collaboration of a venturesome Privacy Infrastructure Provider partner.

At the technical level, we are relying on HTTPS and the Web PKI for transport confidentiality and integrity, an `Authorization: Bearer token` header for authentication between API clients and servers, and a JSON Web Token (JWT) signed with a shared HS256 symmetric key distributed out of band for end-to-end authentication of the Authorized Agent (AA) request to the Covered Business (CB).

Security Model for Beta (spec v.0.6–0.8)

0.6 – Update Data Model, Move to Asymmetric Keys

For the 0.6 milestone, we have two goals:

1. Restructure the data model to move all the necessary information into the signed block of the request, and
2. Move from symmetric keys to asymmetric keys, without changing the format of the signature or what library is used to sign the request (i.e. sticking with JWTs).

The core intuition here is that any information not included in the signed block of the request an Authorized Agent sends to a Covered Business could be modified either by accident or a malicious third party in a way that the Covered Business would not be able to detect, and which could look (at least potentially) like malicious behavior on the part of the Authorized Agent. Including all the information necessary to the request in the signed block ensures that it can't be tampered with or corrupted without detection.

This also allows us to rethink the secret `Authorization` token and associated header used to secure requests *while* preserving their integrity, which significantly simplifies the secret management story for implementers, removes the need for the network itself to know or manage secrets, and enables the network to scale much more easily. With asymmetric keys, verifying the signature on the request body is sufficient to allow the Covered Business or PIP to be assured that it's coming from the named Authorized Agent and that AA asserts that the user's identity has been verified to the standards of the network.

We do still need some way to provide authentication and rate-limiting to requests, especially GET requests, and requiring implementers to validate an asymmetric signature on all requests is burdensome. However, we can use the asymmetric key to bootstrap the generation and sharing of a secret `Authorization` token, allowing Authorized Agents to identify themselves

on future non-sensitive requests without signing the request body. This is not expressed in v0.6 but will be folded into v0.7.

When the Authorized Agent makes signed requests, their identity does need to be included outside the envelope **as well**, so that the verifier knows which key to use to verify the signature, and this external identifier **must** be the same as the AA identity included **inside** the envelope, to prevent a malicious or confused intermediary replaying requests from one AA against a different CB. This AA identifier is the only information which should be sent outside the signed envelope.

The 0.6 protocol spec will include further guidance on JWT ciphers to use, key management best practices, how to effectively move from symmetric HS256 keys to asymmetric RSA keys, and how to verify the request, since getting the details right here is essential to the security of the protocol and the safety of user requests.

The core questions we want to answer from the security side in this release are:

- Are we including the right information in the requests, and can we validate it correctly?
- Does the usage of asymmetric keys and signatures we envision here provide us the guarantees we want it to provide?

We anticipate that feedback from partners and learnings from our own experience building against the spec will deliver the answers we need to proceed to v0.7.

0.7 – Move from JWTs to Libsodium

For the 0.7 milestone, we want to move from using JSON Web Tokens (JWTs) as the data encoding format, key format, and cryptographic library family to using a combined-mode libsodium-signed JSON request.

This choice represents both the best advice from cryptographers and cryptographic engineers we trust, as well as our own experience working with JWT libraries. Libsodium is easier to integrate correctly and supports much stronger and more modern cryptographic algorithms (Ed25519). While a lot of us have substantial experience using JWTs correctly having used them as part of e.g. OAuth2 integrations, anybody who has done so knows exactly how fiddly they can be to get right. Libsodium is a drop-in replacement for JWTs and allows implementers to remove quite a bit of that fiddly code, for example the need to base64url-encode and -decode the tokens, and completely eliminates any of their litany of vulnerabilities.

Each Authorized Agent will need to generate a libsodium signing private and public keypair and distribute the public half to each Covered Business out of band, as well as maintain a mapping to ensure they send CB requests to the right endpoint. On the CB or PIP side, they will need to maintain a mapping from each to its associated public key. (Since these are public keys it's fine to email or Signal them around or whatever at least.)

Libsodium takes care of ciphers, etc. The 0.7 protocol spec will include further guidance on key management best practices and how to verify the request, since getting the details right after the signature has been verified is still essential to the security of the protocol and the safety of user requests.

The core questions we want to answer from the security side for the 0.7 release are:

- How straightforward is it to integrate libsodium? Is it really a drop-in replacement for JWTs? Can we use it to generate asymmetric signatures the way we intend to here? Does using it that way provide us the guarantees we want it to provide?
- Do we understand what information each party (AA, CB, etc) needs to maintain in order to route requests successfully and to verify them correctly? (This will inform the design of the directory services described below.)

As with 0.6, we anticipate that feedback from partners and learnings from our experience building against the 0.7 spec will deliver the answers we need to proceed to v0.8.

0.8 – Incorporate Feedback, Move Towards Servicing Live Consumer Rights Requests

For the 0.8 milestone, we will be incorporating feedback from the previous two milestones, settling on finalized choices for the data model and cryptographic library.

The core question we need to answer from the security side for this release is:

- Are we (AAs, CBs, PIPs, the DRP network) ready to start making, receiving, and actioning real live consumer data rights requests from members of the general public?

We'll answer this by asking consortium members to vote on if the protocol is fit-for-purpose and sufficiently secure for production use.

Security Model North Star (v.0.9–1.3ish)

As mentioned at the beginning, this work brings us two crucial steps closer to a mature security model which is suitable for a system that is trusted by Authorized Agents, Privacy Infrastructure Providers, Covered Businesses, and consumers alike.

Since we believe that the core work is connecting consumers via Authorized Agents with Covered Businesses (which are served by Privacy Infrastructure Providers), ultimately we envision that the DRP network will run a directory service of Covered Businesses which Authorized Agents use to route their requests to the correct endpoint, as well as an Authorized Agent Directory which Covered Businesses use to validate that a request came from a particular Authorized Agent.

This would free the Authorized Agents from needing to know or maintain an understanding of each Covered Businesses' endpoint URLs, and the Covered Businesses from needing to know or manage the Authorized Agents' public signing keys, and instead provide trusted services which each can query to discover the other.

Moving Forward; Feedback?

The work we describe here will be essential to our understanding of DRP's security, and the associated roadmap should enable us to test assumptions quickly and course-correct on our own time and resources, while minimizing thrashing with partners.

If you'd like to discuss either the overall security model North Star or these specific steps and offer comments or critique, we will be holding office hours on November 15th and November 21st at 16:00 UTC.

You can also email the authors of this document—Kevin (kevinr@complexsystems.group) and Ryan (ryan.rix.consultant@consumer.org)—directly. We welcome and look forward to hearing from you!