Network Working Group
Internet-Draft
Intended status: Experimental
Expires: January 3, 2019

S. Li
EFF
C. Man
J. Watson
Stanford University
July 02, 2018

Delegated Distributed Mappings
draft-watson-dinrg-delmap-00

Abstract

Delegated namespaces (domain names, IP address allocation, etc.) underpin almost every Internet entity but are centrally managed, unilaterally revokable, and lack a common interface. This draft specifies a generalized scheme for delegation with a structure that supports explicit delegation guarantees. The resulting data may be secured by any general purpose distributed consensus protocol; clients can query the local state of any number of participants and receive the correct result barring a compromise at the consensus layer.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 3, 2019.

Table of Contents

1. Introduction

Internet entities rely heavily on delegated namespaces to function properly. Typical web services might have been delegated a domain name under which they host the entirety of their public-facing content, or obtain a public IP range from their ISP, acquiring a portion of a namespace originally assigned by an Internet Numbers Registry [RFC7249]. An enormous amount of economic value is therefore placed in these assignments, or mappings, yet they are dangerously ephemeral. Delgating authorities can unilateraly revoke and replace the assignments they've made (maliciously or accidentally), compromising infrastructure security.

Presented in this draft is a generalized mechanism for delegating and managing such mappings. Specifically, we describe the structure for a distributed directory with support for delegation "commitments" that have an explicit duration. Certain known entities are assigned namespaces, loosely associated with a service provided by that entity (i.e domain prefixes for DNS Authorities). Under that namespace, are authorized to create mapping records, or _cells_, a unit of ownership in the service. A namespace's cells are grouped into a logical unit we term a _table_.

Table cells may also explicitly document the delegation of a portion of the authority's namespace to another entity with a given public key, along with a guarantee on that delegation's lifetime. Each delegation forms a new table, for which the delegee is the sole authority. Thus, the delegating entity may not make modifications to a delegated table and need not be trusted by the delegee. The namespace segment may be further delegated to others.

The delegation tables maintain security and consistency through a distributed consensus algorithm. When a participant receives an update, they verify and submit it to the consensus layer, after which, if successful, the change is applied to its associated table. Clients may query any number of trusted servers and expect the result to be correct barring widespread collusion.

The risk of successful attacks on this system vary based on the consensus scheme used. Detailed descriptions of specific protocol implementations are out of scope for this draft, but at a minimum, the consensus algorithm must apply mapping updates in a consistent order, prevent equivocation or unauthorized modification, and enforce the semantic rules associated with each table. We find that federated protocols such as the Stellar Consensus Protocol [I-D.mazieres-dinrg-scp] are promising given their capability for open participation, broad diversity of interests among consensus participants, and a measure of accountability for submitting deceptive updates.

This document specifies the structure for authenticated mapping management and its interface with a consensus protocol implementation.

2. Structure

Trust within the delegation structure is solely based on public key signatures. Namespace authorities must sign any mapping additions, modifications, delegations, and revocations as proof to the other consensus participants that such changes are legitimate. For the sake of completeness, the public key and signature types are detailed below. All types in this draft are described in XDR [RFC4506].

```
typedef publickey opaque<>; /* Typically a 256 byte RSA signature */

struct signature {
    publickey pk;
    opaque data<>;
};
```

2.1. Cells

Cells are the basic unit of the delegation structure. In general, they define an authenticated mapping record that may be queried by clients. We describe two types of cells:

```
enum celltype {
    VALUE = 0,
    DELEGATE = 1
};
```

Value cells store individual mapping entries. They resolve a lookup key to an arbitrary value, for example, an encryption key associated with an email address or a the address of an authoritative nameserver for a given DNS zone. The public key of the cell's owner (e.g. the email account holder, the zone manager) is also included, as well as a signature authenticating the current version of the cell. The cell must be signed either by the "owner_key", or in some cases, the authority of the table containing the cell, as is described below. The cell owner may rotate their public key at any time by signing the transition with the old key.

```
struct valuecell {
    opaque value<>;
    publickey owner_key;
    signature transition_sig; /* Owner or table authority */
};
```

Delegate cells have a similar structure but different semantics. Rather than resolving an individual mapping, they authorize the delegee to create arbitrary value cells within an assigned namespace. This namespace must be a subset of the _delegator_'s own namespace range. The delegee is identified by their public key. Finally, each delegate cell and subsequent updates to the cell are signed by the delegator - this ensures that the delegee cannot unilaterally modify its namespace, which limits the range of mappings they can legitimately create.

```
struct delegatecell {
    opaque namespace<>;
    publickey delegee;
    signature authority_sig;  /* Delegator only */
};
```

Both cell types share a set of common data members, namely a set of UNIX timestamps recording the creation time and, if applicable, the time of last modification. They are useful indicators and will likely be useful in updating consensus nodes that have fallen behind.

An additional "commitment" timestamp must be present in every mapping. It is an explicit guarantee on behalf of the authority creating the cell that the mapping will remain valid until at least the specified time. Therefore, while value cell owners may modify their cell at any moment, the authority cannot successfully change (or remove) the cell until its commitment expires. Similarly, delegated namespaces are guaranteed to be valid until the commitment timestamp. This creates a tradeoff between protecting delegees from arbitrary delegator action and allowing simple reconfiguration that can be customized for the use case.

```
union innercell switch (celltype type) {
case VALUE:
    valuecell vcell;
case DELEGATE:
    delegatecell dcell;
};

struct cell {
    unsigned hyper create_time;     /* 64-bit UNIX timestamps */
    unsigned hyper *revision_time;
    unsigned hyper commitment_time;
    innercell c;
}
```

2.2. Tables

Every cell is stored in a table, which groups all the mappings created by a single authority public key for a specific namespace. Individual cells are referenced by an application-specific label in a lookup table. Below, we allow for a single lookup key to reference a list of cells, for the sake of generality. The combination of a lookup key and a referenced cell value forms a mapping.

```
struct tableentry {
    opaque lookup_key<>;
    cell cells<>;
}
```

Delegating the whole or part of a namespace requires adding a new lookup key for the namespace in question and a matching delegate cell. Each delegation must be validated in the context of the other table entries and the table itself. For example, it should not be possible for the owner of a /8 IPv4 block to delegate the same /16 block to two different delegees. In addition to a collection of entries, each table incorporates a "type" that informs each participating node of the particular delegation rules to apply to table entries.

```
struct table {
    tabletype type;
    tableentry entries<>;
};
```

While there exist more delegation mechanisms than we could reasonably discuss in this draft, we initially propose three general-purpose schemes that cover the majority of use cases:

```
enum tabletype {
    PREFIX = 0,
    SUFFIX = 1,
    FLAT = 2
};
```

The table type informs the validation procedure when performing consensus; all new or updated delegated namespaces must follow the proper format for their table. Prefix-based delegation, such as in an IP delegation use case, requires every table cell value to be prefixed by the table namespace, and no cell value be a prefix of another cell value. Similar rules apply to suffix-based delegation. In cases where arbitrary values may be mapped (e.g. account names for an email service provider), "flat" delegation rules are used.

The delegation rule for a table also determine valid lookup behavior. Given a particular lookup key, "PREFIX"-type tables should have at most one entry whose key is a prefix of the query. Likewise, "SUFFIX" tables have at most one entry whose key is a suffix of the query. As an example, lookup on "irtf.org" in a table of domain names with suffix-based delegation rules may return entries with keys "irtf.org", "tf.org", ".org", etc., but the presence of more than one of these indicates two faulty delegations that control the same namespace.

2.3. Root Key Listing

Each linked group of delegation tables for a particular namespace is rooted by a public key stored in a flat root key listing, which is the entry point for lookup operations. Well-known application identifier strings denote the namespace they control. We describe below how lookups can be accomplished on the mappings.

```
struct rootentry {
    publickey namespace_root_key;
    string application_identifier<>;
    signature listing_sig;
}

struct rootlisting {
    rootentry roots<>;
}
```

A significant open question is how to properly administer entries in this listing, since a strong authority, such as a single root key, can easily protect the listing from spam and malicious changes, but raises important concerns about censorship resilience and potential compromise. A federated approach to management is more in line with the spirit of this draft but opens the door for counter-productive participation. In the "rootentry" description above, we allow for either a root signing key to authenticate mappings, or first-come-first-served self-signed entries. In either case, no more than one key may control the namespace for a specific application identifier.

2.4. Data Structure

Delegation tables are stored in a Merkle hash tree, described in detail in [RFC6962]. In particular, it enables efficient lookups and logarithmic proofs of existence in the tree, and prevents equivocation between different participants. Specifically, we can leverage Google's [Trillian] Merkle tree implementation which generalizes the datastructures used in Certificate Transparency. In map mode, the tree can manages arbitrary key-value pairs at scale. This requires flattening the delegation links such that each table may be queried, while ensuring that a full lookup from the application root be made for each mapping. Given a "rootentry", the corresponding table in the Merkle tree can be found with this concatenation:

```
root_table_name = app_id || namespace_root_key
```

Similarly, tables for delegated namespaces are found at:

```
root_table_name || delegee_key_1 || ... || delegee_key_n
```

Consensus is performed on the Merkle tree containing the flattened collection of tables. While it is possible to reach consensus on entire tables when a cell is modified, this approach does not scale well with the size of the table. Therefore, each table should maintain its entries in its own internal Merkle tree and perform consensus on Merkle proofs for the modified cell.

3. Consensus

Safety is ensured by reaching distributed consensus on the state of the tree. The general nature of a Merkle tree as discussed in the previous section enables almost any consensus protocol to support delegated mappings, with varying guarantees on the conditions under which safety is maintained and different trust implications. For example, a deployment on a cluster of nodes running a classic Byzantine Fault Tolerant consensus protocol such as [PBFT] requires a limited, static membership and can tolerate compromises in up to a third of its nodes. In comparison, proof-of-work schemes including many cryptocurrencies have open membership but rely on economic incentives and distributed control of hashing power to provide safety, and federated consensus algorithms like the Stellar Consensus Protocol (SCP) [I-D.mazieres-dinrg-scp] combine dynamic members with real-world trust relationships but require careful configuration. Determining which scheme, if any, is the "correct" protocol to support authenticated delegation is an open question.

3.1. Validation

Incorrect (potentially malicious) updates to the Merkle tree should be rejected by nodes participating in consensus. Given the limited set of delegation schemes presented in the previous section, each node can apply the same validation procedure without requiring application-specific knowledge. Upon any modification to the tree - addition of a new root entry, table or cell, or modification of an existing cell - the submitted change to the consensus layer should contain:

(1) the updated or newly-created table, and

(2) a Merkle proof containing all the hashes necessary to validate the new root tree hash.

Finally, each node participating in consensus must confirm before voting for the update that:

(1) the Merkle proof is correct, and

(2) an addition to the root key listing is correctly signed by an authorized party, or

(3) for delegate cells:

(3a) a new delegation is correctly authenticated, (3b) the cell contains a valid namespace owned by the delegator, (3c) the

delegation follows the table-specified delegation rules, and (3d) the delegated table is mapped in the Merkle map by the proper key

(4) for value cells:

(4a) a new mapping is correctly authenticated, (4b) the value belongs to the signing authority's namespace, and (4c) does not conflict with other cells in its table

(5) and for all updates, if proposed by the table authority, the cell contains an expired commitment timestamp.

Only after a round of the consensus protocol is successful are the changes exposed to client lookups.

3.2. SCP

While consensus can be reached with many protocols, this section describes how the delegation tables can interface with an SCP implementation.

As discussed above, updates to the delegation tables take the form of Merkle proofs along with the table change itself. Since SCP does not need specific knowledge of the format of these proofs, they directly form the opaque values submitted to the consensus layer. Once a combination of proofs are agreed to as outputs for a given slot, they are applied to the local tables state.

Finally, [I-D.mazieres-dinrg-scp] requires the delegation layer to provide a _validity_ function that is applied to each input value, and a _combining function_ to compose multiple candidate values. For this application, the validity function must implement the logic contained in the previous section. The combining function can simply take the union of the valid proofs proposed by the consensus nodes, rejecting valid, duplicate updates to the same cell in favor of the most up-to-date timestamp.

4. Security Considerations

The security of the delegation tables is primarily tied to the safety properties of the underlying consensus layer. Further, incorrect use of the public key infrastructure authenticating each mapping or compromise of a namespace root key can endanger mappings delegated by the key after their commitments expire.

5. References

5.1. Normative References

[RFC4506] Eisler, M., Ed., "XDR: External Data Representation Standard", STD 67, RFC 4506, DOI 10.17487/RFC4506, May 2006, <https://www.rfc-editor.org/info/rfc4506>.

[Trillian] Google, "Trillian: General Transparency", n.d., <https://github.com/google/trillian>.

5.2. Informative References

[I-D.mazieres-dinrg-scp] Barry, N., Losa, G., Mazieres, D., McCaleb, J., and S. Polu, "The Stellar Consensus Protocol (SCP)", draft-mazieres-dinrg-scp-04 (work in progress), June 2018.

[PBFT] Castro, M. and B. Liskov, "Practical Byzantine Fault Tolerance", 1999, <http://pmg.csail.mit.edu/papers/osdi99.pdf>.

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013, <https://www.rfc-editor.org/info/rfc6962>.

[RFC7249] Housley, R., "Internet Numbers Registries", RFC 7249, DOI 10.17487/RFC7249, May 2014, <https://www.rfc-editor.org/info/rfc7249>.

Acknowledgments

We are grateful for the contributions and feedback on design and applicability by David Mazieres, as well as help and feedback from the IRTF DIN research group, including Dirk Kutscher and Melinda Shore.

This work was supported by The Stanford Center For Blockchain Research.

Authors' Addresses

Sydney Li
Electronic Frontier Foundation
815 Eddy Street
San Francisco, CA 94109
US

Email: sydney@eff.org

Colin Man
Stanford University
353 Serra Mall
Stanford, CA 94305
US

Email: colinman@cs.stanford.edu

Jean-Luc Watson
Stanford University
353 Serra Mall
Stanford, CA 94305
US

Email: jlwatson@cs.stanford.edu