Network Working Group
Internet-Draft
Intended status: Experimental
Expires: August 2, 2019

B. Trammell
C. Fehlmann
ETH Zurich
January 29, 2019

RAINS (Another Internet Naming Service) Protocol Specification
draft-trammell-rains-protocol-05

Abstract

This document defines an alternate protocol for Internet name resolution, designed as a prototype to facilitate conversation about the evolution or replacement of the Domain Name System protocol. It attempts to answer the question: "how would we design DNS knowing what we do now," on the background of a set of properties of an idealized Internet naming service.

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 August 2, 2019.

Copyright Notice

Table of Contents

1. Introduction

This document defines an experimental protocol for providing Internet name resolution services, as a replacement for the Domain Name System (DNS), called RAINS (a recursive acronym expanding to RAINS, Another Internet Naming Service). It is designed as a prototype to facilitate conversation about the evolution or replacement of the Domain Name System protocol, and was developed as a name resolution system for the SCION ("Scalability, Control, and Isolation on Next-Generation Networks") future Internet architecture [SCION]. It attempts to answer the question: "how would we design the DNS knowing what we do now," on the background of the properties of an ideal naming service defined in Section 3.

Its architecture Section 4 is largely compatible with DNS: names in RAINS are organized into zones, associated with authorities, and parts of zones can be delegated to subordinate authorities, just as in DNS; RAINS names follow many of the same conventions as DNS names (with the exception that RAINS is Unicode native, and mandates UTF-8 encoding); and RAINS servers provide services broadly equivalent to recursive, caching, and authoritative DNS servers.

However, its design does take several radical departures from DNS as presently defined and implemented:

- Its information model and data model are separately defined from the protocol, allowing for more natural layering of RAINS messages atop other session and presentation layer protocols than, for example, DNS over HTTPS [RFC8484].

- Delegation from a superordinate zone to a subordinate zone is done solely with cryptography: a superordinate defines the key(s), created by the subordinate, that are valid for signing assertions in the subordinate during a particular time interval. Assertions about names can therefore safely be served from any infrastructure.

- All time references in RAINS are absolute: instead of a time to live, each assertion's temporal validity is defined by the temporal validity of the signature(s) on it.

- All assertions have validity within a specific context. A context is essentially a namespace owned by some authority. The global context, analogous to the root and everything under it in DNS, is a special case of context whose authority is the root itself. Other contexts can be created by any authority under the global context; context is implemented through the rules for chaining signatures to verify validity of an assertion. The use of context explicitly separates global usage of RAINS from local usage thereof, and allows other application-specific naming constraints to be bound to names; see Section 5.2.7. Queries are valid in one or more contexts, with specific rules for determining which assertions answer which queries; see Section 5.5.3.

- There is explicit information about registrars and registrants available in the naming system at runtime: in other words, RAINS integrates parts of the functionality of WHOIS [RFC3912] and RDAP [RFC7482], allowing inline access to registry and registrar information together with naming queries; see Section 5.3.9 and Section 5.3.10.

- Sets of valid characters and rules for valid names are defined on a per-zone basis, and can be verified at runtime; see Section 5.3.6.

- RAINS provides separate namespaces for reverse lookup and a dedicated data type for assertions about addresses, as opposed to rooting reverse lookup in the .arpa top-level domain; see Section 5.2.9.

Instead of using a custom binary framing as DNS, RAINS uses Concise Binary Object Representation (CBOR) [RFC7049], partially in an effort to make implementations easier to verify and less likely to contain potentially dangerous parser bugs [PARSER-BUGS]. As with DNS, CBOR messages can be carried atop any number of substrate protocols. RAINS is presently defined to use TLS over persistent TCP connections (see Section 6). However, the information model was defined to allow the easy defnition of alternate presentations in the future.

1.1. About This Document

The source of this document is available in the repository https://github.com/britram/rains-prototype, and a rendered working copy is available at https://britram.github.io/rains-prototype. Open

issues can be seen and discussed at https://github.com/britram/rains-prototype/issues.

2. Terminology

The terms MUST, MUST NOT, SHOULD, SHOULD NOT, and MAY, when they appear in all-capitals, are to be interpreted as defined in [RFC2119].

In addition, the following terms are used in this document as defined:

- Subject: A name or address about which Assertions can be made.
- Object: A type/value pair of information about a name within an Assertion.
- Assertion: A signed statement about the existence or nonexistence of a mapping from a Subject name and Context to an Object at a given point in time. It is either a Singular Assertion, Shard, or Zone.
- Singular Assertion: A mapping between a Subject and one or several Objects, signed by the Authority for the namespace containing that Subject. See Section 5.2.
- Authority: An entity that has the right to determine which Assertions exist within its Zone
- Delegation: An Assertion proving that an Authority has given the right to make Assertions about names within the part of a namespace identified by a Subject to a subordinate Authority. This subordinate Authority holds a secret key which can generate signatures verifiable using a public key associated with a delegation to the Zone.
- Zone: A portion of a namespace rooted at a given point in the namespace hierarchy. A Zone contains all the Assertions about Subjects tha exist within its part of the namespace.
- Address Assertion: A mapping between a Subject representing an address or address prefix, signed by the Authority for the prefix containing the Subject. See Section 5.2.9.
- Query: An expression of interest in certain types of objects pertaining to a Subject name in one or more contexts. See Section 5.5.

- Context: Additional information about the scope in which an Assertion or Query is valid. See Section 5.2.7 and Section 5.5.3.

- Shard: A group of assertions common to a zone and valid at a given point in time, scoped to a lexicographic range of Subject names within the Zone, for purposes of proving nonexistence of an Assertion. Shards may be encoded to provide either absolute proof or probabilistic assurance of nonexistence. See Section 5.2.2 and Section 5.2.4.

- RAINS Message: Unit of exchange in the RAINS protocol, containing Assertions, Queries, and/or Notifications. See Section 5.1.

- Notification: A RAINS-internal message section carrying information about the operation of the protocol itself. See Section 5.6.

- Authority Service: A service provided by a RAINS Server for publishing Assertions by an Authority. See Section 4.

- Query Service: A service provided by a RAINS Server for answering Queries on behalf of a RAINS Client. See Section 4.

- Intermediary Service: A service provided by a RAINS Server for answering Queries and providing temporary storage for Assertions on behalf of other RAINS Servers. See Section 4.

- RAINS Server: A server that supports the RAINS Protocol, and provides one or more services on behalf of other RAINS Servers and/or RAINS Clients. See Section 4.

- RAINS Client: A client that uses the Query Service of one or more RAINS Servers to retrieve Assertions on behalf of applications that e.g. wish to connect to named services in the Internet.

3. An Ideal Internet Naming Service

We begin by returning to first principles, to determine the dimensions of the design space of desirable properties of an Internet-scale naming service. We recognize that the choices made in the evolution of the DNS since its initial design are only one path through the design space of Internet-scale naming services. Many other naming services have been proposed, though none has been remotely as successful for general-purpose use in the Internet. The following subsections outline the space more generally. It is, of course, informed by decades of experience with the DNS, but identifies a few key gaps which we then aim to address directly with the design of RAINS.

Section 3.1 defines the set of operations a naming service should provide for queriers and authorities, Section 3.2 defines a set of desirable properties of the provision of this service, and Section 3.3 examines implications of these properties.

3.1. Interfaces

At its core, a naming service must provide a few basic functions for queriers that associate the subject of a query with information about that subject. The naming service provides the information necessary for a querier to establish a connection with some other entity in the Internet, given a name identifying it.

- Name to Address: given a Subject name, the naming service returns a set of addresses associated with that name, if such an association exists. The association is determined by the authority for that name. Names may be associated with addresses in one or more address families (e.g. IP version 4, IP version 6). A querier may specify which address families it is interested in. All address families are treated equally by the naming system. This mapping is implemented in the DNS protocol via the A and AAAA RRTYPES.

- Address to Name: given an Subject address, the naming service returns a set of names associated with that address, if such an association exists. The association is determined by the authority for that address. This mapping is implemented in the DNS protocol via the PTR RRTYPE. IPv4 mappings exist within the in-addr.arpa. zone, and IPv6 mappings in the ip6.arpa. zone. These mappings imply a limited set of boundaries on which delegations may be made (octet boundaries for IPv4, nybble boundaries for IPv6).

- Name to Name: given a Subject name, the naming service returns a set of object names associated with that name, if such an association exists. The association is determined by the authority for the subject name. This mapping is implemented in the DNS protocol via the CNAME RRTYPE. CNAME does not allow the association of multiple object names with a single subject, and CNAME may not combine with other RRTYPEs (e.g. NS, MX) arbitrarily.

- Name to Auxiliary Information: given a Subject name, the naming service returns other auxiliary information associated with that name that is useful for establishing communication over the Internet with the entities associated with that name. Most of the other RRTYPES in the DNS protocol implement these sort of mappings.

A naming service also provides other interfaces besides the query interface. The interface it presents to an Authority allows updates to the set of Assertions and Delegations in that Authority's namespace. Updates consist of additions of, changes to, and deletions of Assertions and Delegations. In the present DNS, this interface consists of the publication of a new zone file with an incremented version number, but other authority interfaces are possible.

3.2. Properties

The following properties are desirable in a naming service providing the functions in Section 3.1.

3.2.1. Meaningfulness

A naming service must provide the ability to name objects that its human users find more meaningful than the objects themselves.

3.2.2. Distinguishability

A naming service must make it possible to guarantee that two different names are easily distinguishable from each other by its human users.

3.2.3. Minimal Structure

A naming service should impose as little structure on the names it supports as practical in order to be universally applicable. Naming services that impose a given organizational structure on names will not translate well to societies where that organizational structure is not prevalent.

3.2.4. Federation of Authority

An Authority can delegate some part of its namespace to some other subordinate Authority. This property allows the naming service to scale to the size of the Internet, and leads to a tree-structured namespace, where each Delegation is itself identified with a Subject at a given level in the namespace.

In the DNS protocol, this federation of authority is implemented through delegation using the NS RRTYPE, redirecting queries to subordinate authorities recursively to the final authority. When DNSSEC is used, the DS RRTYPE is used to verify this delegation.

3.2.5. Uniqueness of Authority

For a given Subject, there is a single Authority that has the right to determine the Assertions and/or Delegations for that subject. The unitary authority for the root of the namespace tree may be special, though; see Section 3.2.8.

In the DNS protocol as deployed, unitary authority is approximated by the entity identified by the SOA RRTYPE. The existence of registrars, which use the Extensible Provisioning Protocol (EPP) [RFC5730] to modify entries in the zones under the authority of a top-level domain registry, complicates this somewhat.

3.2.6. Transparency of Authority

A querier can determine the identity of the Authority for a given Assertion. An Authority cannot delegate its rights or responsibilities with respect to a subject without that Delegation being exposed to the querier.

In DNS, the authoritative name server(s) to which a query is delegated via the NS RRTYPE are known. However, we note that in the case of authorities which delegate the ability to write to the zone to other entities (i.e., the registry-registrar relationship), the current DNS provides no facility for a querier to understand on whose behalf an authoritative assertion is being made; this information is instead available via WHOIS. To our knowledge, no present DNS name servers use WHOIS information retrieved out of band to make policy decisions.

3.2.7. Revocability of Authority

An ideal naming service allows the revocation and replacement of an authority at any level in the namespace, and supports the revocation and replacement of authorities with minimal operational disruption.

The current DNS allows the replacement of any level of delegation except the root through changes to the appropriate NS and DS records. Authority revocation in this case is as consistent as any other change to the DNS.

3.2.8. Consensus on Root of Authority

Authority at the top level of the namespace tree is delegated according to a process such that there is universal agreement throughout the Internet as to the subordinates of those Delegations.

3.2.9. Authenticity of Delegation

Given a Delegation from a superordinate to a subordinate Authority, a querier can verify that the superordinate Authority authorized the Delegation.

Authenticity of delegation in DNS is provided by DNSSEC [RFC4033].

3.2.10. Authenticity of Response

The authenticity of every answer is verifiable by the querier. The querier can confirm that the Assertion returned in the answer is correct according to the Authority for the Subject of the query.

Authenticity of response in DNS is provided by DNSSEC.

3.2.11. Authenticity of Negative Response

Some queries will yield no answer, because no such Assertion exists. In this case, the querier can confirm that the Authority for the Subject of the query asserts this lack of Assertion.

Authenticity of negative response in DNS is provided by DNSSEC.

3.2.12. Dynamic Consistency

Consistency in a naming service is important. The naming service should provide the most globally consistent view possible of the set of Assertions that exist at a given point in time, within the limits of latency and bandwidth tradeoffs.

When an Authority makes changes to an Assertion, every query for a given Subject returns either the new valid result or a previously valid result, with known and/or predictable bounds on "how previously". Given that additions of, changes to, and deletions of Assertions may have different operational causes, different bounds may apply to different operations.

The time-to-live (TTL) on a resource record in DNS provides a mechanism for expiring old resource records. We note that this mechanism makes additions to the system propagate faster than changes and deletions, which may not be a desirable property. However, as no context information is explicitly available in DNS, the DNS cannot be said to be dynamically consistent, as different implicitly inconsistent views of an Assertion may be persistent.

3.2.13. Explicit Inconsistency

Some techniques require giving different answers to the same query, even in the absence of changes: the stable state of the namespace is not globally consistent. This inconsistency should be explicit: a querier can know that an answer might be dependent on its identity, network location, or other factors.

One example of such desirable inconsistency is the common practice of "split horizon" DNS, where an organization makes internal names available on its own network, but only the names of externally-visible subjects available to the Internet at large.

Another is the common practice of DNS-based content distribution, in which an authoritative name server gives different answers for the same query depending on the network location from which the query was received, or depending on the subnet in which the end client originating a query is located (via the EDNS Client Subnet extension {RFC7871}}). Such inconsistency based on client identity or network address may increase query linkability (see Section 3.2.18).

These forms of inconsistency are implicit, not explicit, in the current DNS. We note that while DNS can be deployed to allow essentially unlimited kinds of inconsistency in its responses, there is no protocol support for a query to express the kind of consistency it desires, or for a response to explicitly note that it is inconsistent. [RFC7871] does allow a querier to note that it would specifically like the view of the state of the namespace offered to a certain part of the network, and as such can be seen as inchoate support for this property.

3.2.14. Global Invariance

An Assertion which is not intended to be explicitly inconsistent by the Authority issuing it must return the same result for every Query for it, regardless of the identity or location of the querier.

This property is not provided by DNS, as it depends on the robust support of the Explicit Inconsistency property above. Examples of global invariance failures include geofencing and DNS-based censorship ordered by a local jurisdiction.

3.2.15. Availability

The naming service as a whole is resilient to failures of individual nodes providing the naming service, as well as to failures of links among them. Intentional prevention by an adversary of a successful answer to a query should be as hard as practical.

The DNS protocol was designed to be highly available through the use of secondary name servers. Operational practices (e.g. anycast deployment) also increase the availability of DNS as currently deployed.

3.2.16. Lookup Latency

The time for the entire process of looking up a name and other necessary associated data from the point of view of the querier, amortized over all queries for all connections, should not significantly impact connection setup or resumption latency.

3.2.17. Bandwidth Efficiency

The bandwidth cost for looking up a name and other associated data necessary for establishing communication with a given Subject, from the point of view of the querier, amortized over all queries for all connections, should not significantly impact total bandwidth demand for an application.

3.2.18. Query Linkability

It should be costly for an adversary to monitor the infrastructure in order to link specific queries to specific queriers.

DNS over TLS [RFC7858] and DNS over DTLS [RFC8094] provide this property between a querier and a recursive resolver; mixing by the recursive helps with mitigating upstream linkability.

3.2.19. Explicit Tradeoff

A querier should be able to indicate the desire for a benefit with respect to one performance property by accepting a tradeoff in another, including:

- Reduced latency for reduced dynamic consistency
- Increased dynamic consistency for increased latency
- Reduced request linkability for increased latency and/or reduced dynamic consistency
- Reduced aggregate bandwidth use for increased latency and/or reduced dynamic consistency

There is no support for explicit tradeoffs in performance properties available to clients in the present DNS.

3.2.20. Trust in Infrastructure

A querier should not need to trust any entity other than the authority as to the correctness of association information provided by the naming service. Specifically, the querier should not need to trust any intermediary of infrastructure between itself and the authority, other than that under its own control.

DNS provides this property with DNSSEC. However, the lack of mandatory DNSSEC, and the lack of a viable transition strategy to mandatory DNSSEC (see [I-D.trammell-optional-security-not]), means that trust in infrastructure will remain necessary for DNS even with large scale DNSSEC deployment.

3.3. Observations

On a cursory examination, many of the properties of our ideal name service can be met, or could be met, by the present DNS protocol or extensions thereto. We note that there are further possibilities for the future evolution of naming services meeting these properties. This section contains random observations that might inform future work.

3.3.1. Delegation and redirection are separate operations

Any system which can provide the authenticity properties enumerated above is freed from one of the design characteristics of the present domain name system: the requirement to bind a zone of authority to a specific set of authoritative servers. Since the authenticity of a delegation must be protected by a chain of signatures back to the root authority, the location within the infrastructure where an authoritative mapping "lives" is no longer bound to a specific name server. While the present design of DNS does have its own scalability advantages, this implication allows a much larger design space to be explored for future name service work, as a Delegation need not always be implemented via redirection to another name server.

3.3.2. Unicode alone may not be sufficient for distinguishable names

Allowing names to be encoded in Unicode goes a long way toward meeting the meaningfulness property (see Section 3.2.1) for the majority of speakers of human languages. However, as noted by the Internet Architecture Board (see [IAB-UNICODE7]) and discussed at the Locale-free Unicode Identifiers (LUCID) BoF at IETF 92 in Dallas in March 2015 (see [LUCID]), it is not in the general case sufficient for distinguishability (see Section 3.2.2). An ideal naming service may therefore have to supplement Unicode by providing runtime support

for disambiguation of queries and assertions where the results may be indistinguishable.

3.3.3. Implicit inconsistency makes global invariance challenging to verify

DNS does not provide a generalized form of explicit inconsistency, so efforts to verify global invariance, or rather, to discover Assertions for which global invariance does not hold, are necessarily effort-intensive and dynamic. For example, the Open Observatory of Network Interference performs DNS consistency checking from multiple volunteer vantage points for a set of targeted (i.e., likely to be globally variant) domain names; see https://ooni.torproject.org/nettest/dns-consistency/.

4. RAINS Protocol Architecture

The RAINS architecture is simple, and vaguely resembles the architecture of DNS. A RAINS Server is an entity that provides transient and/or permanent storage for assertions about names and addresses, and a lookup function that finds assertions for a given query about a name or address, either by searching local storage or by delegating to another RAINS server. RAINS servers can take on any or all of three roles:

- authority service, acting on behalf of an authority to ensure properly signed assertions are made available to the system (equivalent to an authoritative server in DNS);
- query service, acting on behalf of a client to answer queries with relevant assertions (equivalent to a recursive resolver in DNS), and to validate assertions on the client's behalf; and/or
- intermediary service, acting on behalf of neither but providing storage and lookup for assertions with certain properties for query and authority servers (partially replacing, but not really equivalent to, caching resolvers in DNS).

RAINS Servers use the RAINS Protocol defined in this document to exchange queries and assertions.

From the point of view of an authority (an entity owning some part of the namespace by virtue of holding private keys associated with a zone delegation), the RAINS protocol is used to publish signed assertions toward one or more RAINS servers configured to provide authority service for their domains. Since the signatures on these assertions expire periodically, the authority must publish assertions continuously toward the authority services. In order to provide a

DNS-like operational experience, a RAINS server providing authority service may be colocated with the infrastructure for publishing assertions; however, the architecture of the protocol means these functions need not be colocated.

Clients are configured, or use some out-of-band discovery mechanism, to contact one or more query services using the RAINS protocol, and may trust those services to verify assertion signatures on the client's behalf.

In this way, the same protocol is used between servers, from client to server, and from publisher to server, with minor differences among the interactions implemented as profiles. See Section 6 for details

The protocol itself is designed in terms of its information and data model, detailed in Section 5. Since all RAINS information is carried in messages containing assertions, and an assertion is not valid unless it is signed, the validity of an assertion is separated from whence the assertion was received. This means the RAINS protocol itself is merely a means for moving RAINS assertions around, and moving RAINS queries to places where they can be answered. This document defines bindings for carrying RAINS messages over TLS over TCP, but bindings to other transports (e.g. QUIC [QUIC]) or session layers (e.g. HTTP [RFC7540]) would be trivial to design, and the protocol provides a capability mechanism for discovering alternate transports.

5. Information and Data Model

The RAINS Protocol is based on an information model containing three primary kinds of objects: Assertions, Queries, and Notifications. An Assertion contains some information about a name or address, and a Query contains a request for information about a name or address. Queries are answered with Assertions. Notifications provide information about the operation of the protocol itself. The protocol exchanges RAINS Messages, which act as envelopes containing Assertions, Queries, and Notifications. RAINS Messages also provide for capabilities-based versioning of the protocol, and for recognition of a chunk of CBOR-encoded binary data at rest to be recognized as a RAINS message.

The RAINS data model is a relatively straightforward mapping of the information model to the Concise Binary Object Representation (CBOR) [RFC7049], such that Assertions are split into four subtypes depending on their scope and purpose:

- Singular Assertions and Zones for a positive proof of the existence of an association between a name and an Object;

o Shards and P-Shards for negative proof thereof.

Messages, Singular Assertions, Shards, P-Shards, Zones, Queries, and Notifications are each represented as a CBOR map of integer keys to values, which allows each of these types to be extended in the future, as well as the addition of non-standard, application-specific information to RAINS messages and data items. A common registry of map keys is given in Table 1. RAINS implementations MUST ignore any data objects associated with map keys they do not understand. Integer map keys in the range -22 to +23 are reserved for the use of future versions or extensions to the RAINS protocol, due to the efficiency of representation of these values in CBOR.

Message and Assertion contents, signatures and object values are implemented as type- prefixed CBOR arrays with fixed meanings of each array element; the structure of these lower-level elements can therefore not be extended. Message section types are given in Table 2, object types in Table 4, and signature algorithms in Table 10.

Code	Name	Description
0	signatures	Signatures on a message or section
1	capabilities	Capabilities of server sending message
2	token	Token for referring to a data item
3	subject-name	Subject name in an Assertion, Shard, P-Shard or Zone
4	subject-zone	Zone name in an Assertion, Shard, P-Shard or Zone
5	subject-addr	Subject address in address assertion
6	context	Context of an Assertion, Shard, P-Shard, Zone or Query
7	objects	Objects of an Assertion
8	query-name	Fully qualified name for a Query
9	reserved	Reserved for future use
10	query-types	Acceptable object types for a Query

	11	range	Lexical range of Assertions in Shard or P-Shard
	12	query-expires	Absolute timestamp for query expiration
	13	query-opts	Set of query options requested
	14	current-time	Querier's latest assertion timestamp for a query
	15	reserved	Reserved for future use
	16	reserved	Reserved for future use
	17	query-keyphases	All requested key phases of a Query
	19	reserved	Reserved for future use
	20	assertions	Singular Assertion content of a Shard or Zone
	21	note-type	Notification type
	22	note-data	Additional notification data
	23	content	Content of a Message or a P-Shard

Table 1: CBOR Map Keys used in RAINS

The information model is designed to be representation-independent, and can be rendered using alternate structured-data representations that support the concepts of maps and arrays. For example, YAML or JSON could be used to represent RAINS messages and data structures for debugging purposes. However, signatures over messages and assertions need a single canonical representation of the object to be signed as a bitstream. For RAINS, this is the CBOR representation canonicalized as in Section 5.7.1; therefore alternate representations are always secondary to the CBOR data model.

The following subsections describe the information and data model of a RAINS message from the top down.

5.1. Messages

A Message is a self-contained unit of exchange in the RAINS protocol. Messages have some content (the Assertions, Queries, and/or Notifications carried by the Message) tagged with a token (see

Section 5.8). They may also carry information about peer capabilities, and an optional signature.

More concretely, a Message is represented as a CBOR map with the CBOR tag value 15309736, which identifies the map as a RAINS message. This map MUST contain a token key (2) and a content key (23), and MAY contain a capabilities key (1) a signatures key (0).

The value of the content key is an array of zero or more Message Sections, as defined in Section 5.1.1

The value of the token key is an opaque 16-byte octet array used to link Messages, Queries, and Notifications; see Section 5.8 for details.

The value of the signatures key, when present, is an array of Signatures over the entire Message, generated as in Section 5.7, and to be verified against an infrastructure key (see Section 5.3.11) for the RAINS Server originating the message.

The value of the capabilities key, when present, is an array of Capabilities or the hash thereof. The first Message sent from one peer to another MUST contain the capabilities key. The capabilities mechanism is described in Section 5.9.

5.1.1. Message Section structure

Each Message Section in the Message's content value is represented as a two-element array. The first element in the array is the message section type, encoded as an integer as in Table 2. The second element in the array is a message section body, a CBOR map defined as in the subsections Section 5.2, Section 5.5, and Section 5.6

Code	Name	Description
1	assertion	Singular Assertion (see Section 5.2.1)
-1	revassertion	Address Assertion (see Section 5.2.9)
2	shard	Shard (see Section 5.2.2)
3	p-shard	P-Shard (see Section 5.2.4)
4	zone	Zone (see Section 5.2.3)
5	query	Query (see Section 5.5)
-5	revquery	Address Query (see Section 5.5.4)
23	notification	Notification (see Section 5.6)

Table 2: Message Section Type Codes

5.2. Assertions

Information about names in RAINS is carried by Assertions. An Assertion is a statement about a mapping from a Subject name (or in the case of an Address Assertion, a subject prefix or address) to one or several Object values, signed by some Authority for the namespace containing the Assertion, with a temporal validity determined by the lifetime of the signature(s) on the Assertion.

The subject of an Assertion is identified by a name in three parts:

- the subject zone name, identifying the namespace within which the subject is contained;
- the subject name, identifying the name of the subject within that zone; and
- the subject context, as in Section 5.2.7, identifying the context for purposes of explicit inconsistency.

The types of Objects that can be associated with a Subject are described in Section 5.3.

There are four kinds of Assertions, distinguished by their scope (how many Subjects are covered by a single Assertion) and their utility (whether the Assertion can be used for positive proof of a Subject-

Object association, for negative proof of the lack of such an association, or both):

- Singular Assertions contain a set of Objects associated with a single given subject name in a given zone in a given context. The signature on a Singular Assertion can be used to prove the existence of an association between the subject name and the Objects within the Assertion. Singular Assertions are described in detail in Section 5.2.1.

- Zones contain all Singular Assertions that have the same zone and context values. The signature on a Zone can be used to prove both the existence of an association between a subject name and an Object, as well as the absence of such an association. Zones are described in detail in Section 5.2.3. If signed, the Singular Assertions within a Zone can also be used on their own, as if they were contained within a Message directly; in this case they inherit zone and context information from the containing zone.

- Shards contain Singular Assertions for every Object associated with every subject name in a given lexicographic range of subject names within a given zone in a given context. The signature on a Shard can be used to prove the nonexistence of an Object for a subject name within its range. Shards are described in detail in Section 5.2.2. If a Singular Assertion within a Shard is signed, it inherits zone and context information from the containing shard and can also be used outside the Shard.

- P-Shards (or Probabilistic Shards) contain a data structure that can be used to demonstrate, within predictable bounds of false-negative probability, the nonexistence of an Object for a subject name within a lexicographic range of subject names within a given zone in a given context. They allow an efficiency-accuracy tradeoff for negative proofs. P-Shards are described in detail in Section 5.2.4

5.2.1. Singular Assertions

A Singular Assertion contains a set of Objects associated with a single given subject name in a given zone in a given context. A Singular Assertion with a valid signature can be used as a positive answer to a query for a name. It is represented as a CBOR map. The keys present in this map depend on whether the Singular Assertion is contained in a Message, Shard or Zone.

Singular Assertions contained directly within a Message's content value cannot inherit any values from their containers, and therefore

MUST contain the signatures (0), subject-name (3), subject-zone (4), context (6), and objects (7) keys.

Singular Assertions within a Shard or Zone can inherit values from their containers. A contained Singular Assertion MUST contain the subject-name (3), and objects (7) keys. It MAY contain the signatures (0) key. The subject-zone (4) and context (6) keys MUST NOT be present. They are assumed to have the same value as the corresponding values in the containing Shard or Zone for signature generation and signature verification purposes; see Section 5.7.

The value of the signatures (0) key, if present, is an array of one or more Signatures as defined in Section 5.7. Signatures on a contained Assertion are generated as if the inherited subject-zone and context values are present in the Assertion. The signatures on the Assertion are to be verified against the appropriate key for the Zone containing the Assertion in the given context.

The value of the subject-name (3) key is a UTF-8 encoded [RFC3629] string containing the name of the subject of the assertion. The subject name may cover multiple levels of hierarchy, separated by the '.' character. The fully-qualified name of an Assertion is obtained by joining the subject-name to the subject-zone with a '.' character. The subject-name must be valid according to the nameset expression for the zone, if any (see Section 5.3.6).

The value of the subject-zone (4) key, if present, is a UTF-8 encoded string containing the name of the zone in which the assertion is made and MUST end with '.' (the root zone). If not present, the zone of the assertion is inherited from the containing Shard or Zone.

The value of the context (6) key, if present, is a UTF-8 encoded string containing the name of the context in which the assertion is valid. Both the authority-part and the context-part MUST end with a '.'. If not present, the context of the assertion is inherited from the containing Shard or Zone. See Section 5.2.7 for more.

The value of the objects (7) key is an array of objects, as defined in Section 5.3. Note that this array MAY be empty; see Section 5.2.5.

5.2.2. Shards

A Shard contains Singular Assertions for every Object within a zone in a given context whose subject name falls within a specified lexicographic range. A Shard with a valid signature, within which a subject name should fall (i.e. appearing within that Shard's range), but within which there is no Singular Assertion for the specified

subject name and Object, can therefore be taken as a proof of nonexistence for that subject name and Object. Shards are used exclusively for negative proof; the individual signatures on their contained Singular Assertions are used for positive proof of the existence of an assertion.

The content of a Shard (in terms of the number of Singular Assertions it covers) is chosen by the Authority of the zone for which the Shard is valid. There is an inherent tradeoff between the number of Singular Assertions within a Shard and the size of the Shard, and therefore the size of the Message that must be presented as negative proof. P-Shards ("Probabalistic Shards", see Section 5.2.4) allow a different tradeoff, gaining space efficiency and coverage for a fixed, predictable probability of a false positive (i.e., the possibility that the P-Shard cannot be used to prove the nonexistence of a subject which does not, in fact, exist).

A Shard is represented as a CBOR map. Shards MUST contain the signatures (0), subject-zone (4), context (6), range (11), and assertions (20) keys.

The value of the signatures (0) key is an array of one or more Signatures as defined in Section 5.7. Signatures on the Shard are to be verified against the appropriate key for the Shard in the given context.

The value of the subject-zone (4) key is a UTF-8 encoded string containing the name of the zone in which the Singular Assertions within the Shard are made and MUST end with '.' (the root zone).

The value of the context (6) key is a UTF-8 encoded string containing the name of the context in which the Singular Assertions within the Shard are valid. Both the authority-part and the context-part MUST end with a '.'.

The value of the range (11) key is a two element array of strings or nulls (subject-name A, subject-name B). A MUST lexicographically sort before B. If A is null, the shard begins at the beginning of the zone. If B is null, the shard ends at the end of the zone. The shard MUST NOT contain any Singular Assertions whose subject names are equal to or sort before A, or are equal to or sort after B.

The value of the assertions (20) key is a CBOR array of Singular Assertions as defined in Section 5.2.1. These Singular Assertions MUST be sorted (see Section 5.7.1); the set of allowable Singular Assertions is restricted by the range, as above.

5.2.3. Zones

A Zone contains Singular Assertions for every Object associated with every subject name within a given zone in a given context. A Zone with a valid signature can be used either as a positive answer for a query about a name (when its contained Singular Assertions are not signed), or as a negative answer to prove that a given Object does not exist for a given name.

Organizing Singular Assertions into Zones allows operators of zones with few subject names (e.g., used only for simple web hosting, as is the case with many zones in the current Internet naming system) to minimize signing and zone management overhead.

A Zone is represented as a CBOR map. Zones MUST contain the signatures (0), subject-zone (4), context (6), and assertions (20) keys.

The value of the signatures (0) key is an array of one or more Signatures on the Zone as defined in Section 5.7. Signatures on the Zone are to be verified against the appropriate key for the Zone in the given context.

The value of the subject-zone (4) key is a UTF-8 encoded string containing the name of the Zone which MUST end with '.' (the root zone).

The value of the context (6) key is a UTF-8 encoded string containing the name of the context for which the Zone is valid. Both the authority-part and the context-part MUST end with a '.'. See Section 5.2.7

The value of the assertions (20) key is a CBOR array of Singular Assertions as defined in Section 5.2.1. The CBOR array contains all Singular Assertions of this zone and context and they MUST be sorted (see Section 5.7.1).

5.2.4. P-Shards

Shards (Section 5.2.2) can be used as definitive proof of the nonexistence of a name within a zone. P-Shards serve the same purpose, but offer only a probabilistic guarantee of the nonexistence of a name. Specifically, as they are based on Bloom filters, a subject name which does not in fact exist may appear in the P-Shard; in return for this uncertainty, they offer a much more space-efficient way to demonstrate the nonexistence of an Object for a subject name within the zone and context than Shards do. There is a tradeoff between the size of the bit string storing the Bloom filter,

the number of names covered by the P-Shard, and the false positive error rate. The zone Authority can determine how to weight them.

A P-Shard is represented as a CBOR map. This map MUST contain the signatures (0), subject-zone (4), context (6), and content(23) keys. It MAY contain the range(11) key.

The value of the signatures (0) key is an array of one or more Signatures as defined in Section 5.7. The signatures on the P-Shard are to be verified against the appropriate key for the Zone for which the P-Shard is valid in the given context.

The value of the subject-zone (4) key is a UTF-8 encoded string containing the name of the zone within which the names represented in the P-Shard are contained, and MUST end with '.' (the root zone).

The value of the context (6) key is a UTF-8 encoded string containing the name of the context for which the names represented in the P-Shard are valid. Both the authority-part and the context-part MUST end with a '.'.

The value of the range (11) key, if present, is a two element array of strings or nulls (subject-name A, subject-name B). A MUST lexicographically sort before B. If A is null, the P-Shard begins at the beginning of the zone. If B is null, the P-Shard ends at the end of the zone. The P-Shard MUST NOT be used to check the existence of Assertions about subject names equal to or sort before A, or are equal to or sort after B. If the range (11) key is not present, the P-Shard covers then entire zone.

The value of the content (23) key is a three-element array. The first element identifies the algorithm used for generating the bitstring. The second element identifies the hash function in use for generating the bitstring. The third element contains the bitstring itself, as an octet array. The size of the bitstring must be 0 mod 8.

Table 3 enumerates supported generation algorithms; supported hash functions are given in Section 5.4.

Code	Name	Description
1	bloom-km-12	KM-optimized bloom filter with nh=12
2	bloom-km-16	KM-optimized bloom filter with nh=16
3	bloom-km-20	KM-optimized bloom filter with nh=20
4	bloom-km-24	KM-optimized bloom filter with nh=24

Table 3: P-shard generation algorithms

These datastructures generate a bitstring using a Bloom filter and the Kirsch-Mitzenmacher optimization [BETTER-BLOOM-FILTER].

To add a subject-object mapping for a name to a bloom-km structure, the mapping is first encoded as a four-element CBOR array. The first element is the subject name. The second element is the subject zone. The third element is the subject context. The fourth element is the type code as in Table 4 in Section 5.3. This encoded object is then hashed according to the specified hash algorithm. The hash algorithm's output is then split into two parts of equal length x and y. To obtain the nh indexes into the bitstring, the following equation is used:

- (x + i*y) mod bsl, where bsl is the bitstring length and i ∈ [1,nh]

To add a subject-object mapping, all bits at the calculated indices are set to one. To check wether such a mapping exists, all bits at the calculated indices are checked, and the mapping is taken to be in the filter if all bits are one.

5.2.5. Dynamic Assertion Validity

For a given {subject, zone, context, type} tuple, multiple Singular Assertions can be valid at a given point in time; the union of the object values of all of these Singular Assertions is considered to be the set of valid values for that type at that point in time.

A Singular Assertion's objects array MAY be empty. If the only assertion valid for a given {subject, zone, context} tuple at a given point in time is such a Singular Assertion, this is to be interpreted as a statement that a name exists at that point in time, but that it is not mapped to any object.

5.2.6. Semantic of nonexistence proofs

Shards, P-Shards and Zones can all be used to prove nonexistence during their validity. However, real naming systems are dynamic: an Assertion might be created, altered, expired or revoked during the validity period of a Shard, P-Shard or Zone, leading to an inconsistency. Thus, a section proving nonexistence only captures the state at the point in time when it was signed.

5.2.7. Context in Assertions

Assertion contexts are used to provide explicit inconsistency, while allowing Assertions themselves to be globally valid regardless of the query to which they are given in reply. Explicit inconsistency is the simultaneous validity of multiple sets of Assertions for a single subject name at a given point in time. Explicit inconsistency is implemented by using the context to select an alternate chain of signatures to use to verify the validity of an Assertion, as follows:

- The global context is identified by the special context name '.'. Assertions in the global context are signed by the Authority for the subject zone. For example, assertions about the name 'ethz.ch.' in the global context are only valid if signed by the relevant Authority which is either 'ethz.ch.', 'ch.', or '.' depending on the value of the subject zone of the Assertion.

- A local context is associated with a given Authority. The local context's name is divided into an authority-part and a context-part by a context marker ('cx-'). The authority-part directly identifies the Authority whose key was used to sign the Assertion; Assertions within a local context are only valid if signed by the identified Authority. Authorities have complete control over how the contexts under their namespaces are arranged, and over the names within those contexts. Both the authority-part and the context-part must end with a '.'.

Some examples illustrate how context works:

- For the common split-DNS case, an enterprise could place names for machines on its local networks within a separate context. E.g., a workstation could be named 'simplon.cab.inf.ethz.ch.' within the context 'staff-workstations.cx-inf.ethz.ch.' Assertions about this name would be signed by the Authority for 'inf.ethz.ch.'. Here, the context serves simply as a marker, without enabling an alternate signature chain: note that the name 'simplon.cab.inf.ethz.ch' could at the same time be validly signed in the global context by the Authority over that name to allow external users access this workstation. The local context simply

marks this Assertion as internal. This allows a client making requests of local names to know they are local, and for local resolvers to manage visibility of Assertions outside the enterprise: explicit context makes accidental leakage of both Queries and Assertions easier to detect and avoid.

- Contexts make captive-portal interactions more explicit: a captive portal resolver could respond to a query for a common website (e.g. www.google.ch) with a signed response directed at the captive portal, but within a context identifying the location as well as the ISP (e.g. sihlquai.zurich.ch.cx-starbucks.access.some-isp.net.). This response will be signed by the Authority for 'starbucks.access.some-isp.net.'. This signature achieves two things: first, the client knows the result for www.google.ch is not globally valid; second, it can present the user with some indication as to the identity of the captive portal it is connected to.

Further examples showing how context can be used in queries as well are given in Section 5.5.3 below.

Developing conventions for assertion contexts for different situations will require implementation and deployment experience, and is a subject for future work.

5.2.8. Zone-Reflexive Singular Assertions

A zone may make a Singular Assertion about itself by using the string "@" as a subject name. This facility can be used for any object type, but is especially useful for self-signing root zones, and for a zone to make a subsequent key assertion about itself. If a Singular Assertion for an Object about a zone is available both in the zone itself and in the superordinate zone, the assertion in the superordinate zone will take precedence.

5.2.9. Address Assertions

Address assertions map a subject address or subject address prefix to one or more objects. Address assertions provide the equivalent of reverse DNS (IN PTR) for IPv4 and IPv6 addresses. Information about addresses is stored in a completely separate namespace from information about names, rooted at the prefix containing the entire numberspace for a given address family. An Address Assertion with a valid signature can be used as a positive answer to a query for any address within the subject's prefix. It is represented as a CBOR map.

Assertions about addresses are similar to assertions about names, but keyed by address and restricted in terms of the objects they can contain. An Address Assertion body is a CBOR map which MUST contain the signatures (0), subject-addr (5), and objects (7) keys.

The value of the signatures (0) key is an array of one or more Signatures as defined in Section 5.7. A signature on an Address Assertion can be verified against any public key to which an address delegation assertion exists for a prefix containing the Address Assertion's subject address. This implies that, in contrast to Assertions about names, any authority in the hierarchy may sign Assertions about any address within their prefix, even if they have also delegated part of the prefix to another key.

The value of the subject-addr (5) key is a three element array. The first element of the array is the address family encoded as an object type (see Section 5.3); i.e. 2 for IPv6 addresses and 3 for IPv4 addresses. The second element is the prefix length encoded as an integer, 0-128 for IPv6 and 0-32 for IPv4. The third element is the address, encoded as in Section 5.3.2 or Section 5.3.3.

The value of the objects (7) key is an array of objects, as defined in Section 5.3. If the prefix of the subject-addr value is the maximum prefix length for the address family, then the assertion is a Host Address Assertion, and only the object types redirection, delegation, registrant, and name are valid. Otherwise, it is a Prefix Address Assertion, and only the object types redirection, delegation, and registrant are valid.

Address assertions contain no context, as the context in which they are valid (the global addressing context for the given address family) is implied by the address family of the subject name.

5.3. Object Types and Encodings

Each Object associated with a given subject name in a Singular Assertion (see Section 5.2.1) is represented as a CBOR array, where the first element is the type of the object, encoded as an integer in the following table:

Code	Name	Description	Reference
1	name	name associated with subject	Section 5.3.1
2	ip6-addr	IPv6 address of subject	Section 5.3.2
3	ip4-addr	IPv4 address of subject	Section 5.3.3
4	redirection	name of zone authority server	Section 5.3.4
5	delegation	public key for zone delgation	Section 5.3.5
6	nameset	name set expression for zone	Section 5.3.6
7	cert-info	certificate information for name	Section 5.3.7
8	service-info	service information for srvname	Section 5.3.8
9	registrar	registrar information	Section 5.3.9
10	registrant	registrant information	Section 5.3.10
11	infrakey	public key for RAINS infrastructure	Section 5.3.11
12	extrakey	external public key for subject	Section 5.3.12
13	nextkey	next public key for subject	Section 5.3.13

Table 4: Object type codes

Subsequent elements contain the object content, encoded as described in the respective subsection below.

5.3.1. Name Alias

A name (1) object contains a name associated with a name as an alias. It is represented as a three-element array. The second element is a fully-qualified name as a UTF-8 encoded string. The third type is an array of object type codes for which the alias is valid, with the same semantics as the query-types (9) key in queries (see Section 5.5).

The name type is roughly equivalent to the DNS CNAME RRTYPE.

5.3.2. IPv6 Address

An ip6-addr (2) object contains an IPv6 address associated with a name. It is represented as a two element array. The second element is a byte array of length 16 containing an IPv6 address in network byte order.

The ip6-addr type is roughly equivalent to the DNS AAAA RRTYPE.

5.3.3. IPv4 Address

An ip4-addr (3) object contains an IPv4 address associated with a name. It is represented as a two element array. The second element is a byte array of length 4 containing an IPv4 address in network byte order.

The ip4-addr type is roughly equivalent to the DNS A RRTYPE.

5.3.4. Redirection

A redirection (4) object contains the fully-qualified name of a RAINS authority server for a named zone. It is represented as a two-element array. The second element is a fully-qualified name of an RAINS authority server as a UTF-8 encoded string.

The redirection type is used to point to a "last-resort" server or server from which assertions about a zone can be retrieved; it therefore approximately replaces the DNS NS RRTYPE.

5.3.5. Delegation

A delegation (5) object contains a public key used to generate signatures on assertions in a named zone, and by which a delegation of a name within a zone to a subordinate zone may be verified. It is represented as an 4-element array. The second element is a signature algorithm identifier as in Section 5.7. The third element is a key phase as in Section 5.7. The fourth element is the public key,

formatted as defined in Section 5.7 for the given algorithm identifier and RAINS delegation chain keyspace.

Delegations approximately replace the DNS DNSKEY RRTYPE.

5.3.6. Nameset

A nameset (6) object contains an expression defining which names are allowed and which names are disallowed in a given zone. It is represented as a two- element array. The second element is a nameset expression to be applied to each name element within the zone without an intervening delegation.

The nameset expression is represented as a UTF-8 string encoding a modified POSIX Extended Regular Expression format (see POSIX.2) to be applied to each element of a name within the zone. A name containing an element that does not match the valid nameset expression for a zone is not valid within the zone, and the nameset assertion can be used to prove nonexistence.

The POSIX character classes :alnum:, :alpha:, :ascii:, :digit:, :lower:, and :upper: are available in these regular expressions, where:

- :lower: matches all codepoints within the Unicode general category "Letter, lowercase"
- :upper: matches all codepoints within the Unicode general category "Letter, uppercase"
- :alpha: matches all codepoints within the Unicode general category "Letter".
- :digit: matches all codepoints within the Unicode general category "Number, decimal digit"
- :alnum: is the union of :alpha: and :digit:
- :ascii: matches all codepoints in the range 0x20-0x7f

In addition, each Unicode block is available as a character class, with the syntax :ublkXXXX: where XXXX is a 4 or 5 digit, zero-prefixed hex encoding of the first codepoint in the block. For example, the Cyrillic block is available as :ublk0400:.

Unicode escapes are supported in these regular expressions; the sequence \uXXXX where XXXX is a 4 or 5 digit, possibly zero-prefixed hex encoding of the codepoint, is substituted with that codepoint.

Set operations (intersection and subtraction) are available on character classes. Two character class or range expressions in a bracket expression joined by the sequence && are equivalent to the intersection of the two character classes or ranges. Two character class or range expressions in a bracket expression joined by the sequence - are equivalent to the subtraction of the second character class or range from the first.

For example, the nameset expression:

[[:ublk0400:]&&[:lower:][:digit:]]+

matches any name made up of one or more lowercase Cyrillic letters and digits. The same expression can be implemented with a range instead of a character class:

[\u0400-\u04ff&&[:lower:][:digit:]]+

Nameset expression support is experimental and subject to (radical) change in future revisions of this specification.

5.3.7. Certificate Information

A cert-info (7) object contains an expression binding a certificate or certificate authority to a name, such that connections to the name must either use the bound certificate or a certificate signed by a bound authority. It is represented as an five-element array.

The second element is the protocol family specifier, describing the cryptographic protocol used to connect, as defined in Table 5. The protocol family defines the format of certificate data to be hashed. The third element is the certificate usage specifier as in Table 6, describing the constraint imposed by the assertion. These are defined to be compatible with Certificate Usages in the TLSA RRTYPE for DANE [RFC6698]. The fourth element is the hash algorithm identifier, defining the hash algorithm used to generate the certificate data, as in Table 7. The fifth item is the data itself, whose format is defined by the protocol family and hash algorithm.

Code	Name	Protocol family	Certificate format
0	unspec	Unspecified	Unspecified
1	tls	Transport Layer Security (TLS) [RFC8446]	[RFC5280]

Table 5: Certificate information protocol families

Protocol family 0 leaves the protocol family unspecified; client validation and usage of cert-info assertions, and the protocol used to connect, are up to the client, and no information is stored in RAINS. Protocol family 1 specifies Transport Layer Security version 1.3 [RFC8446] or a subsequent version, secured with PKIX [RFC5280] certificates.

Code	Name	Certificate usage
2	ta	Trust Anchor Certificate
3	ee	End-Entity Certificate

Table 6: Certificate information usage values

A trust anchor certificate constraint specifies a certificate that MUST appear as the trust anchor for the certificate presented by the subject of the Assertion on a connection attempt. An end-entity certificate constraint specifies a certificate that MUST be presented by the subject on a connection attempt.

Certificate information is hashed using an appropriate hash function described in Section 5.4; hash functions are identified by a code as in Table 7. Code 0 is used to store full certificates in RAINS assertions, while other codes are used to store hashes for verification.

For example, in a cert-info object with values [7, 1, 3, 3, (data)], the data would be a 48 SHA-384 hash of the ASN.1 DER-encoded X.509v3 certificate (see Section 4.1 of [RFC5280]) to be presented by the endpoint on a connection attempt with TLS version 1.2 or later.

The cert-info type replaces the TLSA DNS RRTYPE.

5.3.8. Service Information

A service-info (8) object gives information about a named service. Services are named as in [RFC2782]. It is represented as a four-element array. The second element is a fully-qualified name of a host providing the named service as a UTF-8 string. The third element is a transport port number as a positive integer in the range 0-65535. The fourth element is a priority as a positive integer, with lower numbers having higher priority.

The service-info type replaces the DNS SRV RRTYPE.

5.3.9. Registrar Information

A registrar (9) object gives the name and other identifying information of the registrar (the organization which caused the name to be added to the namespace) for organization-level names. It is represented as a two element array. The second element is a UTF-8 string of maximum length 4096 bytes containing identifying information chosen by the registrar, according to the registry's policy.

The protocol does not mandate a format for this string; however, it is RECOMMENDED that authorities place a Registration Data Access Protocol (RDAP) Entity URL or RDAP Entity path segment, as defined in section 3.1.5 of [RFC7482], in the registrar information field. A querier can assume that a URL or an entity path segment (i.e. a string beginning with the substring "entity/") is an RDAP reference to the registrar.

5.3.10. Registrant Information

A registrant (10) object gives information about the registrant of an organization-level name. It is represented as a two element array. The second element is a UTF-8 string wcontaining this information, with a format chosen by the registrant according to the registry's policy.

The protocol does not mandate a format for this string; however, it is RECOMMENDED that authorities place a Registration Data Access Protocol (RDAP) Entity URL or RDAP Entity path segment, as defined in section 3.1.5 of [RFC7482], in the registrant information field. A querier can assume that a URL or an entity path segment (i.e. a string beginning with the substring "entity/") is an RDAP reference to the registrant.

5.3.11. Infrastructure Key

An infrakey (11) object contains a public key used to generate signatures on messages by a named RAINS server, by which a RAINS message signature may be verified by a receiver. It is identical in structure to a delegation object, as defined in Section 5.3.5. Infrakey signatures are especially useful for clients which delegate verification to their query servers to authenticate the messages sent by the query server.

5.3.12. External Key

An extrakey (12) object contains a public key used to generate signatures on assertions in a named zone outside of the normal delegation chain. It is represented as an 4-element array, where the second element is a signature algorithm identifier, and the third element is keyspace identifier, as in Section 5.7. The fourth element is the public key, as defined in Section 5.7 for the given algorithm identifier. An extrakey may be matched with a public key obtained through other means for additional authentication of an assertion.

5.3.13. Next Delegation Public Key

A nextkey (13) object contains the a public key that a zone owner would like its superordinate to delegate to in the future. It is represented as an 6-element array. The second element is a signature algorithm identifier as in Section 5.7. The third element is a key phase as in Section 5.7. The fourth element is the public key, as defined in Section 5.7 for the given algorithm identifier. The fifth element is the requested-valid-since time, and the sixth element is the requested-valid-until time, formatted as for signatures as in Section 5.7. See Section 7.6 for more.

5.4. Hash Functions

Hash algorithms are used in several places in the RAINS data model:

- hashing certificate data in cert-info objects (see Section 5.3.7)
- hashing assertions into Bloom filters and checking if a subject-object mapping within a zone for the given context and type is present in a Bloom filter (see Section 5.2.4)
- hashing Assertion and Message data as part of generating a MAC (see Section 5.7)

Hash functions are identified by a code given in Table 7. The Applicability column determines where in the RAINS Protocol a specific hash function might be used. Applicability "C" means the hash is valid for use in a certificate info object, "P" that it can be used for hashing assertions for P-shards, "S" that it can be used for hashing Assertions and Messages for signatures.

Code	Name	Reference	Length	Applicability
0	nohash	(data not hashed)	var.	C
1	sha-256	[RFC6234]	32	CPS
2	sha-512	[RFC6234]	64	CS
3	sha-384	[RFC6234]	48	CS
4	shake256	[RFC8419]	32	PS
5	fnv-64	[FNV]	8	P
6	fnv-128	[FNV]	16	P

Table 7: Hash algorithms

5.5. Queries

Information about requests for information about names is carried in Queries. A Query specifies the name and object types about which information is requested, information about how long the querier is willing to wait for an answer, and additional options indicating the querier's preferences about how the query should be handled.

In contrast to Singular Assertions, the subject in a Query is given as a fully-qualified name - the subject name concatenated to the zone name with a '.', since a querier may not know the zone name associated with a fully-qualified name.

There are two kinds of queries supported by the RAINS data model:

- Query (or Normal Query): a request for information about one or several types of a given subject, about which the querier expresses no prior information.

- Confirmation Query: a request for information about one or several types of a given subject, for which the querier already has a

valid cached Assertion, but for which the querier would like a new Assertion if available. Confirmation queries are covered in Section 5.5.2.

Both queries are carried in a Query message section. Each Query contained in a Message represents a separate Query.

A Query body is represented as a CBOR map. Queries MUST contain the query-name (8), context (6), query-types (10), and query-expires (12) keys. Queries MAY contain the query-opts (13), query-keyphases (17) keys, and/or current-time (14) keys.

The value of the query-name (8) key is a UTF-8 encoded string containing the name for which the query is issued and MUST end with a '.' (the root zone).

The value of the context (6) key is a UTF-8 encoded string containing the name of the context to which a query pertains. A zero-length string indicates that assertions will be accepted in any context.

The value of the query-types (10) key is an array of integers encoding the type(s) of Objects (as in Section 5.3) acceptable in answers to the query. All values in the query-type array are treated at equal priority: for example, [2,3] means the querier is equally interested in both IPv4 and IPv6 addresses for the query-name. An empty query-types array indicates that objects of any type are acceptable in answers to the query.

The value of the query-expires (12) key is a CBOR integer epoch timestamp identified with tag value 1 and encoded as in section 2.4.1 of [RFC7049]. After the query-expires time, the query will have been considered not answered by the original issuer and can be ignored.

The value of the query-keyphases (17) key, if present, is an array of integers representing all key phases (see Section 5.7) desired in delegation and nextkey answers to queries (see Section 5.3.5 and Section 5.3.13). The value of the query-keyphases key is ignored for all Queries where query-types does not include delegation or nextkey. A query for a delegation or nextkey object that does not contain a query-keyphases key SHOULD return information for all available keyphases.

The value of the query-opts (13) key, if present, is an array of integers in priority order of the querier's preferences in tradeoffs in answering the query. See Section 5.5.1.

The value of the current-time (14) key, if present, is the timestamp of the latest information available at the querier for the queried

subject and object types. See Section 5.5.2 for details of how confirmation queries work.

5.5.1. Query Options

RAINS supports a set of query options to allow a querier to express preferences. Query options are advisory.

Code	Description
1	Minimize end-to-end latency
2	Minimize last-hop answer size (bandwidth)
3	Minimize information leakage beyond first hop
4	No information leakage beyond first hop: cached answers only
5	Expired assertions are acceptable
6	Enable query token tracing
7	Disable verification delegation (client protocol only)
8	Suppress proactive caching of future assertions
9	Maximize freshness of result

Table 8: Query Option Codes

Options 1-5 and 9 specify performance/privacy tradeoffs. Each server is free to determine how to minimize each performance metric requested; however, servers MUST NOT generate queries to other servers if "no information leakage" is specified, and servers MUST NOT return expired Assertions unless "expired assertions acceptable" is specified.

Option 6 specifies that the token on the message containing the query (see Section 5.8) should be used on all queries resulting from a given query, allowing traceability through an entire RAINS infrastructure. The resulting queries SHOULD also carry Option 6. When Option 6 is not present, queries sent by a server in response to an incoming query must use different tokens.

By default, a client service will perform verification on a negative query response and return a 404 No Assertion Exists Notification for queries with a valid and verified proof of nonexistence, within a Message signed by the query service's infrakey. Option 7 disables this behavior, and causes the query service to return the Shard, P-Shard or Zone for verification by the client. It is intended to be used with untrusted query services.

Option 8 specifies that a querier's interest in a query is strictly ephemeral, and that future assertions related to this query SHOULD NOT be proactively pushed to the querier.

Option 9 specifies that the querier would prefer a fresh result to one from the server's cache. If the server is not running an authority service for the queried subject, it can honor this request by issuing a query toward the authority. As this could be used for denial-of-service-attacks, a server honoring Option 9 SHOULD limit the rate of "freshness" queries it issues.

5.5.2. Confirmation Queries

A Query containing a current-time key is a Confirmation Query, used by a server to refresh a cached query result. The querier passes the timestamp of the most recent result it has cached, taken from the most recent start time of the validity of the signature(s) on the Assertion(s) that may answer it. If the answer to a Confirmation Query is not newer than the given timestamp, the server SHOULD answer with a Notification of type 304 (see Section 5.6). Otherwise, the most recent Assertion answering the query is returned.

The value of the current-time key is represented as a CBOR integer epoch timestamp identified with tag value 1 and encoded as in section 2.4.1 of [RFC7049].

5.5.3. Context in Queries

Context is used in Queries as it is in Assertions (see Section 5.2.7). The Context section of a query contains the context of desired Assertions; a special "any" context (represented by the empty string) indicates that Assertions in any context will be accepted. Assertion contexts in an answer to a Query that is not about the "any" context MUST match the context in the Query.

Query contexts can also be used to provide additional information to RAINS servers about the query. For example, context can provide a method for explicit selection of a CDN server not based on either the client's or the resolver's address (see [RFC7871]). Here, the CDN creates a context for each of its content zones, and an external

service selects appropriate contexts for the client based not just on client source address but passive and active measurement of performance. Queries for names at which content resides can then be made within these contexts, with the priority order of the contexts reflecting the goodness of the zone for the client. Here, a context might be 'zrh.cx-cdn-zones.some-cdn.com.' for names of servers hosting content in a CDN's Zurich data center. A client could represent its desire to find content nearby by making queries in the zrh.cx-, fra.cx- (Frankfurt), and ams.cx- (Amsterdam) contexts of the 'cdn-zones.some-cdn.com.' Authority. In all cases, the Assertions themselves will be signed by the Authority for 'cdn-zones.some-cdn.com.', accurately representing that it is the CDN, not the owner of the related name in the global context, that is making the Assertion.

As with assertion contexts, developing conventions for query contexts for different situations will require implementation and deployment experience, and is a subject for future work.

5.5.4. Address Queries

Queries for assertions about addresses are similar to queries for assertions about names, but have semantic restrictions similar to those for Address Assertions. An Address Query body is a map. Queries MUST contain the subject-addr (5), query-types (10), and query-expires (12) keys. Address Queries MAY contain query-opts (13) key.

The value of the subject-addr (5) key is a three element array. The first element of the array is the address family encoded as an object type (see Section 5.3); i.e. 2 for IPv6 addresses and 3 for IPv4 addresses. The second element is the prefix length encoded as an integer, 0-128 for IPv6 and 0-32 for IPv4. The third element is the address, encoded as in Section 5.3.2 or Section 5.3.3.

The value of the query-types (10) key is an array of integers encoding the type(s) of objects (as in Section 5.3) acceptable in answers to the query. All values in the query-type array are treated at equal priority: [4,5] means the querier is equally interested in both redirection and delegation for the subject-addr. An empty query-types array indicates that objects of any type are acceptable in answers to the query. As with Address Assertions, only object types redirection, delegation, registrant, and name are valid on Address Queries.

The value of the query-expires (12) key is a CBOR integer counting seconds since the UNIX epoch UTC, identified with tag value 1 and encoded as in section 2.4.1 of [RFC7049]. After the query-expires

time, the query will have been considered not answered by the original issuer.

The value of the query-opts (13) key, if present, is an array of integers in priority order of the querier's preferences in tradeoffs in answering the query, as in Table 8. See Section 5.5.1 for more.

When answering Address Queries, an Address Assertion with a more-specific prefix is preferred over a less-specific in response to a Address Query.

5.6. Notifications

Notifications contain information about the operation of the RAINS protocol itself. A Notification body is represented as a CBOR map, which MUST contain the token (2) and note-type (21) keys, and MAY contain the note-data (22) key.

The value of the token (2) key is a 16-byte array, which MUST contain the token of the Message to which the Notification is a response. See Section 5.8.

The value of the note-type key is encoded as an integer as in the Table 9.

Code	Description	See Also
100	Connection heartbeat	Section 6.1.2
304	Confirmation query has latest answer	Section 5.5.2
399	Send full capabilities	Section 5.9
400	Bad message received	
403	Inconsistent message received	Section 6.5
404	No assertion exists	Section 6.3
406	Message not acceptable for service	Section 6.3 Section 6.4
413	Message too large	Section 6.1.1
500	Unspecified server error	
504	No assertion available	Section 6.3

Table 9: Notification Type Codes

Note that the status codes are chosen to be mnemonically similar to status codes for HTTP [RFC7231].

The value of the note-data (22) key, if present, is a UTF-8 encoded string with additional information about the notification, intended to be displayed to an administrator to help debug the issue identified by the Notification.

Notification codes 400 and 500 signal error conditions. 400 is a general message noting that a client or server could not parse a message, and 500 notes that the server failed to process a message due to some internal error. Sending these notifications is optional, according to server policy and configuration.

5.7. Signatures

RAINS supports multiple signature algorithms and hash functions for signing Assertions for cryptographic algorithm agility [RFC7696]. A RAINS signature algorithm identifier specifies the signature algorithm; a hash function for generating the HMAC and the format of

the encodings of the signature values in Assertions and Messages, as well as of public key values in delegation objects.

RAINS signatures have five common elements: the algorithm identifier, a keyspace identifier, a key phase, a valid-since timestamp, and a valid-until timestamp. Signatures are represented as an array of these five values followed by additional elements containing the signature data itself, according to the algorithm identifier.

The following algorithms are supported:

Alg ID	Signatures	Hash/HMAC	Format
1	ed25519	sha-512	See Section 5.7.2
2	ed448	shake256	See Section 5.7.2

Table 10: Defined signature algorithms

As noted in Section 5.7.2, support for Algorithm 1, ed25519, is REQUIRED; other algorithms are OPTIONAL.

The keyspace identifier associates the signature with a method for verifying signatures. This facility is used to support signatures on assertions from external sources (the extrakey object type). At present, one keyspace identifier is defined, and support for it is REQUIRED.

Keyspace ID	Name	Signature Verification Algorithm
0	rains	RAINS delegation chain; see Section 5.7

Within the RAINS delegation chain keyspace, the key phase is an unbounded, unsigned integer matching a signature's key phase to the delegation key phase. Multiple keys may be valid for a delegation at a given point in time, in order to support seamless rollover of keys, but only one per key phase and algorithm may be valid at once. The third element of delegation objects and signatures is the key phase.

Valid-since and valid-until timestamps are represented as CBOR integers counting seconds since the UNIX epoch UTC, identified with tag value 1 and encoded as in section 2.4.1 of [RFC7049].

A signature in RAINS is generated over a byte stream representing the data element to be signed. The signing process is defined as follows:

- Render the element to be signed into a canonical byte stream as specified in Section 5.7.1.
- Generate a signature on the resulting byte stream according to the algorithm selected.
- Add the full signature to the signatures array at the appropriate point in the element.

To verify a signature, generate the byte stream as for signing, then verify the signature according to the algorithm selected.

5.7.1. Canonicalization

The byte stream representing a data element over which signatures are generated and verified is a canonicalized CBOR object representing the data element.

Signatures may be attached to any form of Assertion, as well as to Messages as a whole.

First, to canonicalize signature metadata to allow it to be protected by the signature, regardless of the type of data element:

- recursively strip all signatures from the content of the data element.
- add a single-element signatures array at the level in the data structure where the generated signature will be attached, containing the information common to all signatures: the algorithm identifier, a keyspace identifier, a key phase, a valid-since timestamp, and a valid-until timestamp, but omitting any signature content.

Then follow the canonicalization steps below appropriate for the type of data element to be signed:

To generate a canonicalized Singular Assertion:

- sort the objects array by ascending order of object type (Table 4), then by ascending numeric or lexicographic order of each subsequent array element in the object(s)' representation.
- sort the CBOR map by ascending order of its keys (Table 1).

To generate a canonicalized Shard:

- sort the objects array in each Singular Assertion contained in the assertions array as, above.
- sort the assertions array by lexicographic order of the serialized canonicalized byte string representing the assertion. Note that this will cause the assertions array to be sorted in lexicographic order of subject name, as well.
- sort the CBOR map by ascending order of its keys (Table 1).

To generate a canonicalized Zone:

- sort the objects array in each Singular Assertion contained in the assertions array as, above.
- sort the assertions array by lexicographic order of the serialized canonicalized byte string representing the assertion. Note that this will cause the assertions array to be sorted in lexicographic order of subject name, as well.
- sort the CBOR map by ascending order of its keys (Table 1).

To generate a canonicalized P-Shard:

- sort the CBOR map by ascending order of its keys (Table 1).

To generate a canonicalized Message:

- preserve the order of the Message Sections within the Message.
- canonicalize each Section as appropriate by following the canonicalization steps for the appropriate Section type, above.

It is RECOMMENDED that RAINS implementations generate and send only Messages whose contents are sorted according to the canonicalization rules in this section, since the sorting operation is in any case necessary to generate and verify signatures. However, an implementation MUST NOT assume that a Message it receives is sorted according to these rules.

5.7.2. EdDSA signature and public key format

EdDSA public keys consist of a single value, a 32-byte bit string generated as in Section 5.1.5 of [RFC8032] for Ed25519, and a 57-byte bit string generated as in Section 5.2.5 of [RFC8032] for Ed448. The fourth element in a RAINS delegation object is this bit string

encoded as a CBOR byte array. RAINS delegation objects for Ed25519 keys with value k are therefore represented by the array [5, 1, phase, k]; and for Ed448 keys as [5, 2, phase, k].

Ed25519 and Ed448 signatures are are a combination of two non-negative integers, called "R" and "S" in sections 5.1.6 and 5.2.6, respectively, of [RFC8032]. An Ed25519 signature is represented as a 64-byte array containing the concatenation of R and S, and an Ed448 signature is represented as a 114-byte array containing the concatenation of R and S. RAINS signatures using Ed25519 are therefore the array [1, 0, phase, valid-since, valid-until, R|S]; using Ed448 the array [2, 0, phase, valid-since, valid-until, R|S].

Ed25519 keys are generated as in Section 5.1.5 of [RFC8032], and Ed448 keys as in Section 5.2.5 of [RFC8032]. Ed25519 signatures are generated from a normalized serialized CBOR object as in Section 5.1.6 of [RFC8032], and Ed448 signatures as in section 5.2.6 of [RFC8032].

RAINS Server and Client implementations MUST support Ed25519 signatures for delegation.

5.8. Tokens

Messages and Notifications contain an opaque token (2) key, whose content is a 16-byte array, and is used to link Messages to the Queries they respond to, and Notifications to the Messages they respond to. Tokens MUST be treated as opaque values by RAINS servers.

A Message sent in response to a Query (normal and update) MUST contain the token of the Message containing the Query. Otherwise, the Message MUST contain a token selected by the server originating it, so that future Notifications can be linked to the Message causing it. Likewise, a Notification sent in response to a Message MUST contain the token from the Message causing it (where the new Message contains a fresh token selected by the server). This allows sending multiple Notifications within one Message and the receiving server to respond to a Message containing Notifications (e.g. when it is malformed).

Since tokens are used to link Queries to replies, and to link Notifications to Messages, regardless of the sender or recipient of a Message, they MUST be chosen by servers to be hard to guess; e.g. generated by a cryptographic random number generator.

When a server creates a new Query to forward to another server in response to a Query it received, it MUST NOT use the same token on

the delegated query as on the received query, unless option 6 Enable Tracing is present in the received query, in which case it MUST use the same token.

5.9. Capabilities

The capabilities (1) key in a RAINS message allows the sender of that message to communicate its capabilities to its peer. Capabilities MUST be sent on the first message sent from one peer to another.

A peer's capabilities can be represented in one of two ways:

- an array of uniform resource names specifying capabilities supported by the sending server, taken from the table below, with each name encoded as a UTF-8 string.

- a SHA-256 hash of the CBOR byte stream derived from normalizing such an array by sorting it in lexicographically increasing order, then serializing it.

If a peer receives a message from a counterpart for which it does not have the hash of the capabilities, it can ask for the next message to contain a list of these capabilities by sending a message containing notification 399.

This mechanism is inspired by [XEP0115], and is intended to be used to reduce the overhead in exposing common sets of capabilities. Each RAINS server can cache a set of recently-seen or common hashes,

The following URNs are presently defined; other URNs will specify future optional features, support for alternate transport protocols and new signature algorithms, and so on.

URN	Meaning
urn:x-rains:tlssrv	Listens for TLS/TCP connections (see Section 6.1.1

A RAINS server MUST NOT assume that a peer server supports a given capability unless it has received a message containing that capability from that server. An exception are the capabilities indicating that a server listens for connections using a given transport protocol; servers and clients can also learn this information from RAINS itself (given redirection and service-info Assertions for a named zone) or from external configurations.

6. RAINS Protocol

RAINS is a message-exchange protocol based around a CBOR data model. Since CBOR is self-framing - a CBOR parser can determine when a CBOR object is complete at the point at which it has read its final byte - RAINS messages requires no external framing, and can be carried on a variety of transport protocols.

These transport bindings serve to transfer Messages containing Queries toward servers that can answer them, and to transfer Assertions toward clients that have indicated an interest in them. The interpretation and action implied by the arrival of a RAINS Message at a peer is not affected by the transport used to send it.

6.1. Transport Bindings

This document defines one transport binding for RAINS: TLS-over-TCP in Section 6.1.1. Each transport binding offers a different set of tradeoffs. Carrying RAINS Messages over persistent TLS 1.3 (or later) connections [RFC8446] over TCP [RFC0793] protects query confidentiality and integrity while supporting implementation over a ubiquitously-available and well-understood security and transport layer.

6.1.1. TLS over TCP

RAINS servers listen on port 55553 by default. Note that no effort has yet been made to assign this port at IANA; should RAINS be standardized, another port may be chosen. Servers may listen on other TCP ports subject to local configuration. Methods for discovering servers and configuring clients MUST allow for the specification of an alternate port. Servers providing authority service should use service information records (Section 5.3.8) to specify a port on a service name specified by redirection object(s) for the zone; see Section 7.2.

RAINS servers should strive to keep connections open to peer servers, unless it is clear that no future messages will be exchanged with those peers, or in the face of resource limitations at either peer. If a RAINS server needs to send a message to another RAINS server to which it does not have an open connection, it attempts to open a connection with that server.

A RAINS client configured to use one or more servers for query service should strive to keep connections open to those servers.

RAINS servers MUST accept Messages over TCP up to 65536 bytes in length, but MAY accept messages of greater length, subject to

resource limitations of the server. A server with resource limitations MUST respond to a message rejected due to length restrictions with a notification of type 413 (Message Too Large). A server that receives a type 413 notification must note that the peer sending the message only accepts messages smaller than the largest message it's successfully sent that peer, or cap messages to that peer to 65536 bytes in length.

Since a singular assertion with a single Ed25519 signature requires on the order of 180 bytes, it is clear that many full zones won't fit into a single minimum maximum-size message. Authorities are therefore encouraged to publish zones grouped into shards that will fit into 65536-byte messages, to allow servers to reply using these shards when full-zone transfers are not possible due to message size limitations.

6.1.2. Heartbeat Messages

TCP connections between RAINS clients and servers may be associated with in-network state (such as firewall pinholes and/or network address translation cache entries) with relatively short idle timeouts. RAINS provides a simple heartbeat mechanism to refresh this state for long-running connections.

A RAINS peer may send its peer a 100 Connection Heartbeat notification at any time. This message is ignored by the receiving peer.

6.2. Protocol Dynamics

This section illustrates how the RAINS protocol works with one possible set of rules for handling incoming messages and sending outgoing messages as a RAINS server; however, the actions here and the sequence in which they are applied are meant only as one possibility for implementors, and are not normative.

6.2.1. Message Processing

Once a transport connection is established, any server may validly send a message with any content to any other server. A client may send messages containing queries to servers, and a server may sent messages containing anything other than queries to clients.

Upon receipt of a message, a server or client attempts to parse it. If the server or client cannot parse the message at all, it returns a 400 Bad Message notification to the peer. This notification may have a null token if the token cannot be retrieved from the message.

If the server or client can parse the message, it:

- notes the token on the message to send on any message generated in reply to the message.

- processes any capabilities present, replacing the set of capabilities known for the peer with the set present in the message. If the present capabilities are represented by a hash that the server does not have in its cache, it prepares a notification of type 399 ("Capability hash not understood") to send to its peer.

- splits the contents into its constituent message sections, and verifies that each is acceptable. Specifically, queries are not accepted by clients (see Section 6.3), and 404 No Assertion Exists notifications are not accepted by servers. If a message contains an unacceptable section, the server or client returns a 406 Message Not Acceptable for Service notification to its peer, and ceases processing of the message.

It then processes each sections acoording to the rules below.

On receipt of an Assertion (Singular Assertion, Shard, P-Shard, or Zone) section, a server:

- verifies its consistency (see Section 6.5). If the section is not consistent, it prepares to send a notification of type 403 Inconsistent Message to the peer, and discards the section. Otherwise, it:

- determines whether it answers an outstanding query; if so, it prepares to forward the section to the server that issued the query.

- determines whether it is likely to answer a future query, according to its configuration, policy, and query history; if so, it caches the section.

On receipt of an Assertion (Singular Assertion, Shard, P-Shard, or Zone) section, a client:

- determines whether it answers an outstanding query; if so, it considers the query answered. It then:

- determines whether it is likely to answer a future query, according to its configuration, policy, and query history; if so, it caches the section.

On receipt of a query, a server:

1. determines whether it has expired by checking the query-expires value. If so, it drops the query silently. If not, it

2. determines whether it has at least one stored assertion containing a positive answer to the query. If so, it checks to see if the assertion is newer than the current-time value in the query, if present. If the assertion is not newer, it prepares to send a notification of type 304 ("Querier Has Latest Answer") to the peer. Otherwise, it prepares a message containing the stored assertion(s) positively answering the query. If no positive assertion is available, it

3. checks to see whether it has at least one stored proof of nonexistence (shard or p-shard) for the query. If so, it prepares a message containing the negative proof to the peer. It prefers P-Shards to Shards for reasons of efficiency, but must verify that any P-shard does indeed function as a negative proof before sending it.

4. determines whether it has other non-authoritative servers it can forward the query to, according to its configuration and policy, and in compliance with any query options (see Section 5.5.1). If so, it prepares to forward the query to those servers, noting the reply for the received query depends on the replies for the forwarded query. If not, it:

5. determines the responsible authority servers for the zone containing the query name in the query for the context requested, and forwards the query to those authority servers, noting the reply for the received query depends on the reply for the forwarded query.

Query options (see Section 5.5.1) change this handling. If query option 4 ("cached answers only") is set, steps 4 and 5 above are skipped, and the server returns a 504 ("No Assertion Available") notification instead. If query option 9 ("Maximize Freshness") is set, the server might forward a query even if it has a cached answer.

If query delegation fails to return an answer within the maximum of the valid-until time in the received query and a configured maximum timeout for a delegated query, the server prepares to send a 504 No assertion available response to the peer from which it received the query.

When a server creates a new query to forward to another server in response to a query it received, it does not use the same token on

the delegated query as on the received query, unless option 6 ("Enable Tracing") is present in the received query, in which case it does use the same token. The Enable Tracing option is designed to allow debugging of query processing across multiple servers.

When a server creates a new query to forward to another server in response to a query it received, and the received query contains a query-expires time, the delegated query MUST NOT have a query-expires time after that in the received query. If the received query contains no query-expires time, the delegated query MAY contain a query- expires time of the server's choosing, according to its configuration.

On receipt of a notification, a server's behavior depends on the notification type:

- For type 100 "Connection Heartbeat", the server does nothing: these null messages are used to keep long-lived connections open in the presence of network behaviors that may drop state for idle connections.

- For type 399 "Capability hash not understood", the server prepares to send a full capabilities list on the next message it sends to the peer.

- For type 504 "No assertion available", the server checks the token on the message, and prepares to forward the assertion to the associated query.

- For type 413 "Message too large" the server notes that large messages may not be sent to a peer and tries again, or logs the error along with the note-data content.

- For type 400 "Bad message", type 403 "Inconsistent message", type 406 "not supported for service", or type 500 "Server error", the server logs the error along with the note-data content, as these notifications generally represent implementation or configuration error conditions which will require human intervention to mitigate.

On receipt of a notification, a client's behavior depends on the notification type:

- For type 100 "Connection Heartbeat", the client does nothing, as above.

- For type 304 "Querier has newest assertion", the client notes that its cache is up-to-date for the given query.

- For type 399 "Capability hash not understood", the client prepares to send a full capabilities list on the next message it sends to the peer.

- For type 404 "No assertion exists", the client takes the query to be unanswerable. It may reissue the query with query option 7 to do the verification of nonexistence again, if the server from which it received the notification is untrusted.

- For type 413 "Message too large" the client notes that large messages may not be sent to a peer and tries again, or logs the error along with the note-data content.

- For type 400 "Bad message", type 403 "Inconsistent message", type 406 "not acceptable for service", of type 500 "Server error", the client logs the error along with the note-data content, as these notifications generally represent implementation or configuration error conditions which will require human intervention to mitigate.

The first message a server or client sends to a peer after a new connection is established SHOULD contain a capabilities section, if the server or client supports any optional capabilities. See Section 5.9.

If the server is configured to keep long-running connections open, due to the presence of network behaviors that may drop state for idle connections, it sends a message containing a type 100 Connection Heartbeat notification after a configured idle time without any messages containing other content being sent.

6.2.2. Message Transmission

As noted in Section 6.2.1 many messages are sent in reply to messages received from peers. Servers may also originate messages on their own, based on their configuration and policy:

- Proactive queries to retrieve assertions, shards, and zones for which all signatures have expired or will soon expire, for cache management purposes.

- Proactive push of assertions, shards, and zones to other servers, based on query history or other information indicating those servers may query for the assertions they contain.

6.3. Client Protocol

The protocol used by clients to issue queries to and receive responses from a query service is a subset of the full RAINS protocol, with the following differences:

- Clients only process assertion, shard, zone, and notification sections; sending a query to a client results in a 406 Unacceptable notification.

- Clients never listen for connections via TCP; a client must initiate and maintain a transport session to the query server(s) it uses for name resolution.

- Servers only process query and notification sections when connected to clients; a client sending assertions to a server results in a 406 Unacceptable notification.

Since signature verification is resource-intensive, clients delegate signature verification to query servers by default. The query server signs the message containing results for a query using its own key (published as an infrakey object associated with the query server's name), and a validity time corresponding to the signature it verified with the longest lifetime, stripping other signatures from the reply. This behavior can be disabled by a client by specifying query option 7, allowing the client to do its own verification.

6.4. Publication Protocol

The protocol used by authorities to publish assertions to an authority service is a subset of the full RAINS protocol, with the following differences:

- Servers only process assertion, shard, zone, and notification sections when connected to publishers; sending a query to a server via the publication procotol results in a 406 Unacceptable notification. Servers only process notifications for capability negotiation purposes (see Section 5.9).

- Publishers only process notification sections; sending a query or assertion to a publisher results in a 406 Unacceptable notification.

6.5. Enforcing Assertion Consistency

The data model used by the RAINS protocol allows inconsistent information to be asserted, all resulting from misconfigured or

misbehaving authority servers. The following types of inconsistency are possible:

- A Zone omits an Assertion which has the same validity start time as said Assertion.
- A Shard omits an Assertion within its range which has the same validity start time as said Assertion.
- A P-Shard with a given validity start time proves nonexistence of an Assertion with the same validity start time.
- An Assertion prohibited by its Aone's nameset has the same validity start time as the prohibiting nameset Assertion.
- A zone contains a valid reflexive assertion of a given object type with the same validity start time as a valid assertion of the same type for the same name within a supordinate zone, but with a different object value.
- Delegations to more than one key are simultaneously valid for a given context, zone, signature algorithm, and key phase.

RAINS relies on runtime consistency checking to mitigate inconsistency: each server receiving an assertion, shard, or zone SHOULD, subject to resource constraints, ensure that it is consistent with other information it has, and if not, discard all inconsistent assertions, shards, and zones in its cache, log the error, and send a 403 Inconsistent Message to the source of the message.

For RAINS to work in a highly dynamic environment, some time-bounded inconsistencies are allowed to occur. On the one hand, the authority wants to prove nonexistence of a name for a duration of time to make caching possible to reduce query latency and reduce load on its naming servers. On the other hand, the authority would like the flexibility to issue new assertions about previously nonexistent names without waiting for a previous negative proof to expire. Therefore, the defintions of inconsistency above are strictly limited to identical (and therefore non-orderable) validity start times.

7. Operational Considerations

The following subsections discuss issues that must be considered in any deployment of RAINS at scale.

7.1. Discovering RAINS servers

A client that will not do its own verification must be able to discover the query server(s) it should trust for resolution. There are three broad approaches to this discovery process: (1) static client configuration; (2) server configuration as part of dynamic host interface configuration, such as DHCP or provisioning domains; (3) discovery of a RAINS server as an optional service, for example using mDNS. Integration with any of these approaches is

In any case, clients MUST provide a configuration interface to allow a user to specify (by address or name) and/or constrain (by certificate property) a preferred/trusted query server. This would allow client on an untrusted network to use an untrusted locally-available query server to discover a preferred query server (doing key verification on its own for bootstrapping), before connecting to that query server for normal name resolution.

Servers providing query and intermediate service also discover other intermediate servers through static configuration, or through an external, unspecified discovery protocol.

Servers providing query and intermediate service discover servers providing authority service as in Section 7.2, below.

7.2. Bootstrapping RAINS Services

At startup, a server performing recursive lookup MUST have access to at least one of each of these three assertion types: a self-signed delegation assertion of the root zone, a redirection assertion containing the name of an authoritative root name server, and an ip4 or ip6 assertion of the root name server mentioned in the redirection assertion. These assertions must be obtained through a secure out of band mechanism. For a caching server, it is sufficient to have a connection to a recursive resolver which does the lookup on its behalf.

When a zone authority delegates a part of its namespace to a subordinate, it MUST sign and serve the assertions of the three above mentioned types. This information is necessary for a recursive resolver to determine in a recursive lookup where to ask for a more specific answer and to validate the response.

7.3. Cooperative Delegation Distribution

Regardless of any other configuration directive, a RAINS server MUST be prepared to provide a full chain of delegation assertions from the appropriate delegation root to the signature on any assertion it

gives to a peer or a client, whether as additional assertions on a message answering a query, or in reply to a subsequent query. This property allows RAINS servers to maintain a full delegation tree.

7.4. Assertion Lifetime Management

An assertion can contain multiple signatures, each with a different lifetime. Signature lifetimes are equivalent to a time to live in the present DNS: authorities should compute a new signature for each validity period, and make these new signatures available when old ones are expiring.

Since assertion lifetime management is based on a real-time clock expressed in UTC, RAINS servers MUST use a clock synchronization protocol such as NTP [RFC5905].

RAINS servers MAY coalesce assertion lifetimes, e.g. using only the most recent valid-until time in their cache management. This implies that an assertion with valid signatures in time intervals (T1, T2) and (T3, T4) such that T3 > T2 may be cached during the interval (T2, T3) as well. Authorites MUST NOT rely on non-caching or non-availability of assertions during such intervals.

7.5. Secret Key Management

The secret keys associated with public keys for each RAINS server (via infrakey objects) must be available on that server, whether through a hardware or software security device, so they can sign messages on demand; this is particularly important for query servers. In addition, the secret keys associated with TLS certificates for each server (published via certinfo objects) must be available as well in order to establish TLS sessions.

However, storing zone secret keys (associated via delegation objects) on RAINS servers would represent a more serious operational risk. To keep this from being necessary, authority servers have an additional signer interface, from which they will accept and cache any assertion, shard, or zone for which they are authority servers until at least the end of validity of the last signature, provided the signature is verifiable.

7.6. Public Key Management

As signature lifetime is used to manage assertion lifetime, and key rotation strategies may be used both for revocation as well as operational flexibility purposes, RAINS presents a much more dynamic key management environment than that presented by DNSSEC.

7.6.1. Key Phase and Key Rotation

Each signature and public key in a RAINS message is associated with a key phase, allowing multiple keys to be valid for a given authority at any given time. For example, given two key phases and a key validity interval of one day, a phase 0 key would be valid from 00:00 on day 0 to 00:00 on day 1, and a phase 1 key valid from 12:00 on day 0 to 12:00 on day 1. When the phase 0 key expires, it would be replaced by a new phase 0 valid from 00:00 on day 1 to 00:00 on day 2, and so on.

Since the end time of the validity of a signature on an assertion is the maximum of the validity of the signatures on each of the delegations in the delegation chain from the root, key rotation avoids mass expiration of assertions, at the cost of requiring one valid signatures per key phase on at least all delegation assertions. Key rotation schedules are a matter of authority operational policy, but key validity intervals should be longer the closer in the delegation chain an assertion is to the root.

7.6.2. Next Key Assertions

Another problem this dyanmic envrionment raises is how a zone authority communicates to its superordinate that it would like to begin using a new public key to sign its assertions.

This can be done out of band, using private APIs provided by the superordinate authority. Through the nextkey object type, RAINS provides a way for a future public key to be shared with the superordinate authority (and all other queriers) in-band. An authority that wishes to use a new key publishes a reflexive nextkey assertion (i.e., in its own zone, with subject @) with the new public key and a requested valid-since and valid-until time range. The superordinate issues periodic queries for nextkey assertions from its subordinate zone, or the subordinate pushes these assertions to an intermediate service designated to receive them. When the superordinate receives a nextkey, and it decides it wants to delegate to the new key, it creates and signs a delegation assertion.

This process is not mandatory: the superordinate is free to ignore the request, or to use a different time range, depending on its policy and/or the status of its business relationship with the subordinate. The subordinate can discover this, in turn, using its own RAINS queries, or through the delegation assertions being similarly pushed to a designated intermediate service.

8. Experimental Design and Evaluation

The protocol described in this document is intended primarily as a prototype for discussion, though the goal of the document is to specify RAINS completely enough to allow independent, interoperable implementation of clients an servers. The massive inertia behind the deployment of the present domain name system makes full deployment as a replacement for DNS unlikely. Despite this, there are some criteria by which the success of the RAINS experiment may be judged:

First, deployment in simulated or closed networks, or in alternate Internet architectures such as SCION, allows implementation experience with the features of RAINS which DNS lacks (signatures as a first-order delegation primitive, support for explicit contexts, explicit tradeoffs in queries, runtime availability of registrar/ registrant data, and nameset support), which in turn may inform the specification and deployment of these features on the present DNS.

Second, deployment of RAINS "islands" in the present Internet alongside DNS on a per-domain basis would allow for comparison between operational and implementation complexity and efficiency and benefits derived from RAINS' features, as information for future development of the DNS protocol.

9. Security Considerations

This document specifies a new, experimental protocol for Internet name resolution, with mandatory integrity protection for assertions about names built into the information model, and confidentiality for query information protected on a hop-by-hop basis.

9.1. Integrity and Confidentiality Protection

Assertions are not valid unless they contain at least one signature that can be verified from the chain of authorities specified by the name and context on the assertion; integrity protection is built into the information model. The infrastructure key object type allows keys to be associated with RAINS servers in addition to zone authorities, which allows a client to delegate integrity verification of assertions to a trusted query service (see Section 6.3).

Since the job of an Internet naming service is to provide publicly-available information mapping names to information needed to connect to the services they name, confidentiality protection for assertions is not a goal of the system. Specifically, the information model and the mechanism for proving nonexistence of an assertion is not designed to provide resistance against zone enumeration.

On the other hand, confidentiality protection of query information is crucial. Linking naming queries to a specific user can be nearly as useful to build a profile of that user for surveillance purposes as full access to the clear text of that client's communications [RFC7624]. In this revision, RAINS uses TLS to protect communications between servers and between servers and clients, with certificate information for RAINS infrastructure stored in RAINS itself. Together with hop-by-hop confidentiality protection, query options, proactive caching, default use of non-persistent tokens, and redirection among servers can be used to mix queries and reduce the linkability of query information to specific clients.

10. IANA Considerations

The present revision of this document has no actions for IANA.

The authors have registered the CBOR tag 15309736 to identify RAINS messages in the CBOR tag registry at https://www.iana.org/assignments/cbor-tags/cbor-tags.xhtml.

RAINS servers currently listen for connections from other servers by default on TCP port 7753. This port has not been registered with IANA, and is intended only for experimentation with RAINS on closed, non-Internet-connected networks. Future revisions of this document may specify a different port, registered with IANA via Expert Review [RFC5226].

The urn:x-rains namespace used by the RAINS capability mechanism in Section 5.9 may be a candidate for replacement with an IANA-registered namespace in a future revision of this document.

11. Acknowledgments

Thanks to Daniele Asoni, Laurent Chuat, Markus Deshon, Ted Hardie, Joe Hildebrand, Tobias Klausmann, Steve Matsumoto, Adrian Perrig, Raphael Reischuk, Wendy Seltzer, Andrew Sullivan, and Suzanne Woolf for the discussions leading to the design of this protocol, and the definition of an ideal naming service on which it is based. Thanks especially to Stephen Shirley for detailed feedback.

12. References

12.1. Normative References

[FIPS-186-3]
NIST, ., "Digital Signature Standard FIPS 186-3", June 2009.

[FNV] Fowler, G., Noll, L., Vo, K., Eastlake, D., and T. Hansen, "The FNV Non-Cryptographic Hash Algorithm", draft-eastlake-fnv-16 (work in progress), December 2018.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <https://www.rfc-editor.org/info/rfc2782>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7482] Newton, A. and S. Hollenbeck, "Registration Data Access Protocol (RDAP) Query Format", RFC 7482, DOI 10.17487/RFC7482, March 2015, <https://www.rfc-editor.org/info/rfc7482>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8419] Housley, R., "Use of Edwards-Curve Digital Signature Algorithm (EdDSA) Signatures in the Cryptographic Message Syntax (CMS)", RFC 8419, DOI 10.17487/RFC8419, August 2018, <https://www.rfc-editor.org/info/rfc8419>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

12.2. Informative References

[BETTER-BLOOM-FILTER]
Adam Kirsch, . and . Michael Mitzenmacher, "Building a Better Bloom Filter", May 2008.

[I-D.ietf-dprive-dns-over-tls]
Zi, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over TLS", draft-ietf-dprive-dns-over-tls-09 (work in progress), March 2016.

[I-D.ietf-dprive-dnsodtls]
Reddy, T., Wing, D., and P. Patil, "Specification for DNS over Datagram Transport Layer Security (DTLS)", draft-ietf-dprive-dnsodtls-15 (work in progress), December 2016.

[I-D.trammell-optional-security-not]
Trammell, B., "Optional Security Is Not An Option", draft-trammell-optional-security-not-01 (work in progress), January 2019.

[IAB-UNICODE7]
IAB, ., "IAB Statement on Identifiers and Unicode 7.0.0", n.d., <https://www.iab.org/documents/correspondence-reports-documents/2015-2/iab-statement-on-identifiers-and-unicode-7-0-0/>.

[LUCID] Freytag, A. and A. Sullivan, "LUCID problem (slides, IETF 92 LUCID BoF)", n.d., <https://www.ietf.org/proceedings/92/slides/slides-92-lucid-0.pdf>.

[PARSER-BUGS]
Bratus, S., Patterson, M., and A. Shubina, "The Bugs We Have To Kill (USENIX login)", August 2015.

[QUIC] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-18 (work in progress), January 2019.

[RFC3912] Daigle, L., "WHOIS Protocol Specification", RFC 3912, DOI 10.17487/RFC3912, September 2004, <https://www.rfc-editor.org/info/rfc3912>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <https://www.rfc-editor.org/info/rfc4033>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

[RFC5730] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", STD 69, RFC 5730, DOI 10.17487/RFC5730, August 2009, <https://www.rfc-editor.org/info/rfc5730>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <https://www.rfc-editor.org/info/rfc6698>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC7624] Barnes, R., Schneier, B., Jennings, C., Hardie, T., Trammell, B., Huitema, C., and D. Borkmann, "Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement", RFC 7624, DOI 10.17487/RFC7624, August 2015, <https://www.rfc-editor.org/info/rfc7624>.

[RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <https://www.rfc-editor.org/info/rfc7696>.

[RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <https://www.rfc-editor.org/info/rfc7858>.

[RFC7871] Contavalli, C., van der Gaast, W., Lawrence, D., and W. Kumari, "Client Subnet in DNS Queries", RFC 7871, DOI 10.17487/RFC7871, May 2016, <https://www.rfc-editor.org/info/rfc7871>.

[RFC8094] Reddy, T., Wing, D., and P. Patil, "DNS over Datagram Transport Layer Security (DTLS)", RFC 8094, DOI 10.17487/RFC8094, February 2017, <https://www.rfc-editor.org/info/rfc8094>.

[RFC8484] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, DOI 10.17487/RFC8484, October 2018, <https://www.rfc-editor.org/info/rfc8484>.

[SCION] Barrera, D., Reischuk, R., Szalachowski, P., and A. Perrig, "SCION Five Years Later - Revisiting Scalability, Control, and Isolation Next-Generation Networks (arXiv:1508.01651v1)", August 2015.

[XEP0115] Hildebrand, J., Saint-Andre, P., Troncon, R., and J. Konieczny, "XEP-0115 Entity Capabilities", February 2008.

Authors' Addresses

Brian Trammell
ETH Zurich
Universitaetstrasse 6
Zurich 8092
Switzerland

Email: ietf@trammell.ch

Christian Fehlmann
ETH Zurich

Email: fehlmannch@gmail.com