

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 27, 2011

J. Dickinson
Sinodun Internet Technologies
S. Morris
Internet Systems Consortium
R. Arends
Nominet UK
October 24, 2010

Design for a Nameserver Control Protocol
draft-dickinson-dnsop-nameserver-control-01.txt

Abstract

This document presents a design for a nameserver control protocol (NSCP).

A common data model for describing the configuration and operation of a basic, but usable, generic name server is defined. This is expressed in a formal modeling language (YANG) and can be used as the basis of a set of NETCONF operations and capabilities.

The data model described is extensible and will allow for the creation of additional capabilities, ensuring that the protocol can support all the features of a name server.

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 <http://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 April 27, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | | |
|-------------|--|----|
| 1. | Introduction | 4 |
| 1.1. | Background | 4 |
| 1.2. | NSCP | 4 |
| 1.3. | Use of NETCONF | 4 |
| 1.3.1. | NETCONF Overview | 4 |
| 1.3.2. | Why NETCONF? | 5 |
| 1.4. | Requirements notation | 6 |
| 2. | Object Models | 7 |
| 2.1. | NSCP Base Capability | 7 |
| 2.1.1. | Server | 8 |
| 2.1.2. | Statistics | 8 |
| 2.1.3. | DNSSEC Policy | 9 |
| 2.1.4. | Peer | 10 |
| 2.1.5. | Peers | 11 |
| 2.1.6. | Panorama | 11 |
| 2.1.7. | View | 12 |
| 2.1.8. | Access Control List | 13 |
| 2.1.9. | Zone | 15 |
| 2.2. | NSCP Basic Control Capability | 15 |
| 2.2.1. | Methods | 16 |
| 2.3. | NSCP Start Control Capability | 16 |
| 2.3.1. | Methods | 16 |
| 3. | Examples | 17 |
| 3.1. | Obtaining Configuration Information | 17 |
| 3.2. | Modifying Configuration Information | 18 |
| 3.3. | Controlling the Name Server | 20 |
| 4. | IANA Considerations | 21 |
| 5. | Security Considerations | 22 |
| 6. | References | 23 |
| 6.1. | Normative References | 23 |
| 6.2. | Informative | 23 |
| Appendix A. | Meeting the Requirements | 24 |
| A.1. | Management Architecture Requirements | 24 |
| A.1.1. | Expected Deployment Scenarios | 24 |
| A.1.2. | Name Server Types | 25 |

| | | |
|--------------------|--|----|
| A.2. | Management Operation Types | 25 |
| A.2.1. | Control Requirements | 25 |
| A.2.2. | Configuration requirements | 25 |
| A.2.3. | Monitoring Requirements | 26 |
| A.2.4. | Alarm and Event Requirements | 26 |
| A.2.5. | Security Requirements | 26 |
| A.2.6. | Other Requirements | 27 |
| Appendix B. | NSCP Capabilities | 28 |
| B.1. | The NSCP Base Capability | 28 |
| B.1.1. | Capability Name | 28 |
| B.1.2. | Overview | 28 |
| B.1.3. | Dependencies | 28 |
| B.1.4. | Capability Identifier | 28 |
| B.1.5. | New Operations | 28 |
| B.2. | The NSCP Basic Control Capability | 28 |
| B.2.1. | Capability Name | 28 |
| B.2.2. | Overview | 28 |
| B.2.3. | Dependencies | 29 |
| B.2.4. | Capability Identifier | 29 |
| B.2.5. | New Operations | 29 |
| B.3. | The NSCP Start Control Capability | 29 |
| B.3.1. | Capability Name | 29 |
| B.3.2. | Overview | 29 |
| B.3.3. | Dependencies | 29 |
| B.3.4. | Capability Identifier | 29 |
| B.3.5. | New Operations | 29 |
| Appendix C. | YANG Data Model for Base NSCP capability | 30 |
| Appendix D. | YANG Data Model for the NSCP Basic Control Capability | 36 |
| Appendix E. | YANG Data Model for the NSCP Start Control Capability | 37 |
| Authors' Addresses | | 38 |

1. Introduction

1.1. Background

Management of DNS name servers is currently carried out via vendor-specific control, configuration and monitoring methods. Organizations run multiple name server implementations from a variety of vendors. A common method of name server management can simplify administration and reduce cost.

The requirements for the management of name servers have been established and documented [I-D.ietf-dnsop-name-server-management-reqs]. In essence, the document describes a set of common operations that name servers are known to implement.

1.2. NSCP

NSCP is a name server control protocol that meets the requirements set out in [I-D.ietf-dnsop-name-server-management-reqs]. Based around NETCONF [RFC4741], NSCP consists of a common data model for describing the configuration and operation of a basic, generic name server that is comparable with a subset of the features of existing name server implementations. This data model, expressed in a formal modeling language (YANG [RFC6020]), is used as the basis for a set of NETCONF operations and capabilities.

The basic NSCP data model is extensible and allows for the creation of additional NETCONF capabilities that will ensure the protocol can support all the features of a name server.

Using NSCP, a suitable client should be able to communicate with and manage any name server implementing the protocol.

It is the intention of NSCP that the NSCP data model will be transformable into a data model suitable for use with a particular name server implementation. In the longer term it is hoped that name servers will implement the NSCP data model directly.

1.3. Use of NETCONF

1.3.1. NETCONF Overview

NETCONF is a protocol for the management and configuration of network devices, where operations are layered on top of a remote procedure call interface. A client establishes a session with a server via a secure, connection-oriented transport mechanism (such as SSH).

The operations sent to the server, the replies from it and the configuration data itself are encoded in XML.

Within NETCONF, capabilities identify sets of functionality that a client or server may implement. During the setup of the session, client and server exchange a list capabilities. As each system ignores capabilities that it does not require or understand, the two systems can settle on a common set understood by both. These capabilities allow the client and server to agree on

- o New operations
- o Modifications to existing operations
- o The data model(s) being used.

NETCONF provides a set of simple operations that allow management of configuration data and retrieval of state information.

1.3.2. Why NETCONF?

There are a number of reasons for using NETCONF as the basis of NSCP:

- o It is an established application protocol, allowing reuse of existing tools and code.
- o It is connection-oriented, running over secure links. This matches anticipated use of NSCP.
- o Use of XML allows established transformation methods such as XSLT to be used to transform the NSCP configuration in to a vendor specific configuration format.
- o Configuration information is opaque to NETCONF, allowing any data to be transported over it.
- o It supports seamless extension of functionality via its capabilities feature.

It is capabilities that make NETCONF particularly suitable for use as the basis of NSCP. In particular, three requirements of [I-D.ietf-dnsop-name-server-management-reqs] (section 5.1) are:

- o The management solution must be flexible and be able to accommodate new future operations.
- o It must be possible for vendors to extend the standardized management model with vendor-specific extensions.

- o It must be possible for a management station to understand which parts of returned data are specific to a given vendor or other standardized extension.

Capabilities clearly fit these requirements; by defining extensions to NSCP - either vendor-specific or standard ones defined at a later date - as NETCONF capabilities, additional features can be added to NSCP whilst maintaining backwards compatibility with existing systems.

1.4. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Object Models

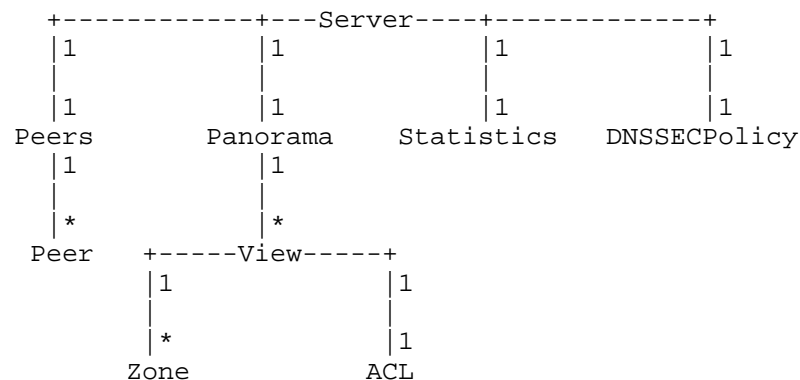
NSCP is built up from a set of capabilities. These provide different parts of the object model as described below.

2.1. NSCP Base Capability

[I-D.ietf-dnsop-name-server-management-reqs] (section 2.1.5) states that a common data model **MUST** be defined. This is a very necessary requirement, since only by establishing a common definition about what is being managed can management clients and name servers exchange meaningful information.

During the design of NSCP, several existing name server implementations were examined and the common details abstracted into a common model. Inevitably, many detailed features of individual name servers are not included. However, the capability mechanism of NETCONF will allow them to be added as vendor-specific extensions.

The following figure shows the overall structure of the object model within the "basic" capability.



(The numbers indicate the cardinality of the associations.) The following sections describe each object in more detail. For every object, the following information is provided:

- o A discussion of the object - what it represents and its purpose.
- o A list of the object's elements - the name of each element and the contents.

2.1.1.1. Server

The server object is the root of the configuration tree and serves as the focus for server-wide operations and repository for server-wide information.

2.1.1.2. Statistics

In the base capability it is assumed that the server is capable of generating the subset of Bind 8 style statistics currently supported by both NSD and Bind 9.

Additional statistics will be the subject of future capabilities.

- o RR
Count of responses received.
- o RNXD
Count of NXDomain received
- o RDupR
Count of duplicate responses
- o RFail
Count of SERVFAIL responses received
- o RFErr
Count of FORMERR responses received
- o RErr
Count of other errors received
- o RAXFR
Count of AXFR initiated
- o RLame
Count of lame delegations received
- o SSysQ
Count of NS address fetches
- o SAns
Count of answers sent
- o SFwdQ
Count of queries sent

- o SDupQ
Count of queries retried
- o RQ
Count of requests received
- o SNXD
Count of NXDomain sent
- o RUQ
Count of non-recursive queries rejected
- o RURQ
Count of recursive queries rejected
- o RUXFR
Count of XFR rejected
- o RUUpd
Count of updates rejected

2.1.3. DNSSEC Policy

The DNSSEC policy defines a policy for the DNSSEC validation and signing operations performed by the name server. Any signing policy will be held in a key and signing policy database (KASP). The details of KASP are still being developed; for now the data model will just specify the location of the database as a string. Trusted keys used for validation will be stored in a manner defined by the implementation, therefore there will be no need for NSCP to specify a trusted keys file. It is also assumed that all name servers managed using NSCP will be DNSSEC-capable.

2.1.3.1. Elements

secure

The name of a domain that must validate as secure. There may be more than one of these.

nonsecure

The name of a domain that may validate as insecure. There may be more than one of these.

KASP-database

The name of the KASP database.

2.1.4. Peer

2.1.4.1. Discussion

A Peer is an object representing an external system or systems (since there is no way to tell if a key or IP address refers to a single system). A Peer is either a system that participates in the nameserver service by being a master or a slave, or is a system that uses the nameserver service. It is identified by a name and must contain either a key element or an address element, or both. All references to external systems in the rest of the NSCP object model refer to a Peer.

2.1.4.2. Elements

- o name
A name for this Peer. This name is unique, and is used elsewhere in the model as a reference to this object.
- o address
Describes an IP address. There can be zero or more address elements in the Peer although, if there are no address elements, a key element **MUST** be present. The contents of the address element are:
 - * ip
This is mandatory and holds the IP address of the Peer. The IP address can be either IPV4 or IPV6. The address/network is represented in standard form (e.g. 192.0.2.0/24 or 2001:0DB8::/32).
 - * port
Optional port number.
- o key
A TSIG key to be used when talking to this Peer. This is optional. The contents of a key element are one of:
 - * hmac-md5
This holds the secret for HMAC-MD5
 - * hmac-sha1
This holds the secret for HMAC-SHA1

Use of an element per algorithm will allow easy augmentation with future algorithms.

A Peer allows for the identification of an external systems. Where

only an address element is present, the identification is clear. Should only a key element be present an external system corresponds to the Peer if it presents a suitable signature. The combination of address and key elements allows TSIG transactions to be more discriminating; an external system corresponds to the Peer if and only if it presents correct signatures and its address is correct.

2.1.5. Peers

A container for Peer objects. This may be useful as it provides a point at which a partial lock can be placed [RFC5717] to lock a group of Peers without affecting the rest of the server.

It is thought that there may be some use for named groups of peers to be allowed. This will be considered more in future versions of this draft.

2.1.5.1. Example

The following is an example of a group of Peers.

```
<peers>
  <peer>
    <name>Server123</name>
    <address>
      <ip>192.0.2.0/24</ip>
      <port>53</port>
    </address>
    <address>
      <ip>2001:0DB8::08:EF</ip>
      <port>53</port>
    </address>
  </peer>
  <peer>
    <name>OtherServer</name>
    <key>
      <hmac-shal>hQdEr7iwwB8ATMuZAJlYFQ==</hmac-shal>
    </key>
  </peer>
</peers>
```

2.1.6. Panorama

2.1.6.1. Discussion

The Panorama is a collection of views, it provides a point at which a partial lock can be placed [RFC5717] to lock all views without

affecting the rest of the server.

2.1.7. View

2.1.7.1. Discussion

The view object can be considered as a virtual server. It groups zones with similar access characteristics. It is more than just the association of a zone and an ACL: a view allows the server to send different replies to clients according to the following criteria

- o Source address (and port).
- o Destination address (and port).
- o Whether or not the query requests recursion.

The replies can differ in terms of

- o Which zones are available to a given client.
- o The contents of those zones.
- o Is recursion available?
- o Is validation performed?
- o Any configuration parameters of the server or zone.

To aid in the definition of a common object model, a view will always exist in a NSCP name server configuration, even if the name server concerned does not implement views. All implementations of NSCP MUST implement a view named "_default".

In this data model all zones in a given view will have the same masters and slaves.

2.1.7.2. Elements

- o name
A name for the view.
- o listen-on
What IP address and port to bind this view to. This can occur zero or more times.
- o recursion
Is recursion enabled in this view ("on" or "off").

- o validation
Is DNSSEC validation performed in this view ("on" or "off").

2.1.8. Access Control List

Access to services on the name server are controlled by Access Control Lists (ACL). Using common nomenclature, an ACL comprises one or more Access Control Entries (ACEs). Each ACE links an attribute of the accessor (an "identifier") to one or more access rights to the resource being controlled.

An ACL is attached to a view. However, it may be useful for it to be attached to a zone. This will be considered further in a future version of this draft.

An ACL contains the following elements:

- o ace
Zero or more access control entries, each entry defining a match between an identifier and access modes.
- o default
Zero or one default access control entries defining access rights should no other ACE match.

The order of ACEs within an ACL is important. When checking for access, the system MUST attempt to match each ACE in turn. If none match, the system MUST attempt to match a default ACE (a wildcard that matches any accessor). If there is no default accessor, access MUST be denied.

2.1.8.1. ace

An ACE links an identifier with one or more access modes. An ACE has the following sub-elements:

- o identifier
This element - of which there must be one and only one - defines what is accessing the system. The element's value is the name of a Peer defining one or more external systems.
- o access
Access elements define what access rights the accessor has to the server, such as operations they are able to undertake and data they are able to see. There are one or more access elements in each ACE. The value of this element is one of the following:

| | |
|--------------|--|
| none | No access. This overrides any other type of access, and denies access to the accessor. |
| notify | Causes the server to take notice of NOTIFY messages from the accessor. |
| nonrecursive | Allows the accessor to make a non-recursive request to the server. |
| recursive | Allows the accessor to make a recursive request to the server. |
| transfer | Allows the accessor to transfer data from the server (either AXFR or IXFR). |
| update | Causes the server to accept dynamic update messages from the accessor. |

2.1.8.2. default

A default access control entry is consulted only if all explicit ACEs fail to match. It does not contain an "identifier" clause (being deemed to match all identifiers), only access elements as described above.

2.1.8.3. Example

The following ACL would allow any accessor to query the zones in this view. Any system in the Peer LocalGroup would also be able to AXFR and IXFR from zones in the view. Systems in the Peer MasterSystem (which would presumably contain a "key" element defining a TSIG/SIG0 key) could dynamically update the zone.

```
<acl>
  <ace>
    <identifier type="peer">LocalGroup</identifier>
    <access>transfer</access>
    <access>nonrecursive</access>
  </ace>
  <ace>
    <identifier type="peer">MasterSystem</identifier>
    <access>update</access>
    <access>nonrecursive</query>
  </ace>
  <default>
    <access>nonrecursive</query>
  </default>
```

</acl>

2.1.9. Zone

2.1.9.1. Discussion

The zone object represents a zone served by the name server and comprises a collection of resource records. In virtually all cases, the zone will not be created by an NSCP client, being defined instead by the contents of a zone file, an external database or by zone transfer. If it is necessary to define zone contents using NSCP then the zone contents can be defined using a Zone Data Capability which will be the subject of a separate draft. This means that with just the base capability it will only be possible to provision zones via AXFR (In other words, to create the zone configuration specifying a master server from which the name server will have to obtain the zone data).

In this data model it is expected that filenames and directory structure will be fixed by the implementation. Therefore, a zone does not have a filename element.

2.1.9.2. Elements

- o name
the name of the zone being served. The name of a zone MUST be unique within a view, although different views may have zones with the same name.
- o master
The identifier of a Peer that can act as a master server for this zones. This can occur zero or more times.
- o slave
The identifier of a Peer that can act as slave server for this zones. This can occur zero or more times. Note: This says nothing about whether or not those servers should be sent notifies. Both master and slave can appear for a single zone.
- o notify
The identifier of a Peer to which notifies should be sent to for this zone. This can occur zero or more times.

2.2. NSCP Basic Control Capability

This capability provides the basic control functions for the operation of the name server. It adds the following methods

2.2.1. Methods

- o server-stop
Stops the server software.
- o server-reload
Reloads the zone data
- o server-restart
stops and the restarts the server software.

2.3. NSCP Start Control Capability

This capability provides an additional control functions. It has been added as a separate capability because they will only be possible if the name server is being managed by an external NSCP process. It adds the following methods:

2.3.1. Methods

- o server-start
Starts the server software.

3. Examples

This section gives some examples of NSCP requests.

3.1. Obtaining Configuration Information

To obtain configuration information from a remote nameserver via NSCP, a NETCONF <get-config> operation is used. <get-config> is used to retrieve all or part of a configuration. By default it uses subtree filtering to select the part of the configuration tree to return. XPath filtering can also be used if the :xpath capability is supported.

A <get-config> is required to specify a source configuration data store. By default NETCONF only specifies the "running" configuration. However, NSCP implementations are free to add additional data stores and advertise their-presence via implementation-specific capabilities (e.g. a "candidate" configuration, see [RFC4741] section 8.3).

The following example request uses <get-config> (in an implementation of NSCP with the :xpath capability) to select a Peer (named "master") from the configuration.

```
<rpc message-id="101">
  <get-config>
    <source>
      <running/>
    </source>
    <filter type="xpath" select="server/peers/peer[name='master']"/>
  </get-config>
</rpc>
```

... and an example of a possible reply is:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <peer xmlns="urn:ietf:nscp:1.0">
      <name>master</name>
      <address>
        <ip>192.0.2.0/24</ip>
        <port>53</port>
      </address>
      <address>
        <ip>2001:0DB8::08:EF</ip>
        <port>53</port>
      </address>
    </peer>
  </data>
</rpc-reply>
```

3.2. Modifying Configuration Information

Modification of configuration information is achieved via `<edit-config>`, another standard operation defined by NETCONF [RFC4741]. It takes an operation attribute that allows the specification of how the target should be modified; elements can be added or removed, or the contents of the `<edit-config>` message can be merged into the target.

The next example updates the NSCP configuration to enable recursion in a view called `my_view`. The "replace" attribute ensures that recursion is enabled, whatever the current state.

```
<rpc message-id="102">
  <edit-config>
    <target>
      <running/>
    </target>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <server xmlns="urn:ietf:nscp:1.0">
        <panorama>
          <view>
            <name>myview</name>
            <recursion xc:operation="replace">on</recursion>
          </view>
        </panorama>
      </server>
    </config>
  </edit-config>
</rpc>
```

On success, the reply looks like:

```
<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

(This is the form of reply expected when the server is merely acknowledging the completion of an NSCP command and not returning data. In the following examples, such replies are omitted.)

In the next example, the NSCP configuration is modified to add a new zone to a secondary. The master is configured to allow AXFR. To configure a master server one would also use a zone data capability to upload or point NSCP at the zone data.

```
<rpc message-id="103">
  <edit-config>
    <target>
      <running/>
    </target>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <server xmlns="urn:ietf:ns:1.0">
        <panorama>
          <view>
            <name>_default</name>
            <zone xc:operation="create">
              <name>example.org</name>
              <master>peer1</master>
              <notify>peer4</notify>
            </zone>
          </view>
        </panorama>
      </server>
    </config>
  </edit-config>
</rpc>
```

The next example is the same as the previous one (i.e. the addition of a zone), but illustrates a server that also supports the :rollback-on-error capability. By advertising this capability, the server guarantees that if the requested change fails for some reason, the existing configuration will be left unaltered; there will be no partial alteration of the configuration. The :rollback-on-error capability is one of several capabilities defined in [RFC4741]

```
<rpc message-id="104">
  <edit-config>
    <target>
      <running/>
    </target>
    <error-option>rollback-on-error</error-option>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <server xmlns="urn:ietf:ncsp:1.0">
        <panorama>
          <view>
            <name>_default</name>
            <zone xc:operation="create">
              <name>example.org</name>
              <master>peer1</master>
              <notify>peer4</notify>
            </zone>
          </view>
        </panorama>
      </server>
    </config>
  </edit-config>
</rpc>
```

3.3. Controlling the Name Server

The following example illustrates how to stop a server if it has the Basic Control capability

```
<rpc message-id="105">
  <server-stop/>
</rpc>
```

With the Start Control capability, it is possible to start a stopped server

```
<rpc message-id="106">
  <server-start/>
</rpc>
```

4. IANA Considerations

 This memo includes no request to IANA.

5. Security Considerations

To be completed.

6. References

6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4741] Enns, R., "NETCONF Configuration Protocol", RFC 4741, December 2006.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.
- [RFC5717] Lengyel, B. and M. Bjorklund, "Partial Lock Remote Procedure Call (RPC) for NETCONF", RFC 5717, December 2009.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

6.2. Informative

- [I-D.ietf-dnsop-name-server-management-reqs]
Hardaker, W., "Requirements for Management of Name Servers for the DNS",
draft-ietf-dnsop-name-server-management-reqs-04 (work in progress), June 2010.

Appendix A. Meeting the Requirements

This section discusses how NSCP meets the requirement described in [I-D.ietf-dnsop-name-server-management-reqs]. The requirements are listed in the same order as those described in that document.

A.1. Management Architecture Requirements

A.1.1. Expected Deployment Scenarios

Nothing in NSCP restricts the range of deployment scenarios. Indeed, the very nature of NETCONF lends itself to supporting different scenarios through the use of capabilities.

A.1.1.1. Zone Size Constraints

Nothing in NSCP restricts the size or number of zones served by any NSCP managed name server.

A.1.1.2. Name Server Discovery

This is not supported in the base NSCP capability. However, nothing prevents this being added later, either as some kind of capability or via a different protocol.

A.1.1.3. Configuration Data Volatility

Nothing in NSCP restricts the frequency of changes to the name server configuration

A.1.1.4. Protocol Selection

It is anticipated that NSCP, along with extensions built using capabilities, can meet the needs of a full management protocol. There may however, be occasions where there is an existing protocol better suited for carrying a particular task. For example, loading zone contents via AXFR/IXFR.

A.1.1.5. Common Data Model

A common data model is a core part of NSCP and is described in detail in this document.

A.1.1.6. Operational Impact

It is not anticipated that NSCP will add significant overhead to any DNS service.

A.1.2. Name Server Types

NSCP, along with extensions built using capabilities will support all the name server types.

A.2. Management Operation Types

A.2.1. Control Requirements

A.2.1.1. Needed Control Operations

NSCP along with the two control capabilities described in this document can provide the minimum set of control operations. Additional operations can be added via new capabilities.

A.2.1.2. Asynchronous Status Notifications

[RFC5277] describes an asynchronous message notification delivery service for NETCONF. This is an optional capability built on the base NETCONF definition. It is expected that this will form part of NSCP and will be considered further in future versions of this draft.

A.2.2. Configuration requirements

A.2.2.1. Served Zone Modification

The base NSCP protocol will be able to add, modify and delete the configuration about served zones. The ability to configure a zone with data will be the subject of additional capabilities described in other drafts. It is anticipated that there may be a variety of zone data modification capabilities to reflect the variety of methods possible for sending zone data to a name server. These may include capabilities that define methods to obtain zone data using DDNS, AXFR, HTTP, SCP, or even over NETCONF itself.

A.2.2.2. Trust Anchor Management

The base NSCP protocol will be able to add, modify and delete trust anchors.

A.2.2.3. Security Expectations

The base NSCP capability will be able to configure validation policies.

A.2.2.4. TSIG Key Management

The base NSCP protocol is able to add, modify and delete peers, which can be identified by TSIG keys.

A.2.2.5. DNS Protocol Authorization Management

NSCP contains a sophisticated access control mechanism that will allow control of

- o Access to operations on zone data.
- o Access to zone data from certain sources.
- o Access to specific DNS protocol services.

A.2.3. Monitoring Requirements

It remains to be decided how much monitoring will form part of the base NSCP capability. Some minimal health monitoring and statistics gathering are included in the base NSCP capability. Future capabilities are expected to allow more state to be monitored.

A.2.4. Alarm and Event Requirements

As mentioned before, [RFC5277] describes an asynchronous message notification delivery service for NETCONF. It is expected that this will form part of NSCP and will be considered further in future versions of this draft.

A.2.5. Security Requirements

A.2.5.1. Authentication

Authentication will be provided by the NETCONF transport layer.

A.2.5.2. Integrity Protection

Integrity Protection will be provided by the NETCONF transport layer.

A.2.5.3. Confidentiality

Confidentiality will be provided by the NETCONF transport layer.

A.2.5.4. Authorization

Authorization will be provided by NETCONF or a capability.

A.2.5.5. Solution Impacts on Security

It is believed that NSCP does minimize any security risks introduced to the name server system. The security is provided by the NETCONF transport layer and a variety of suitable transports are available including ssh.

A.2.6. Other Requirements

A.2.6.1. Extensibility

NSCP is flexible and be able to accommodate new future management operations.

A.2.6.1.1. Vendor Extensions

NSCP does allow vendors to extend the standardized management model with vendor-specific extensions

A.2.6.1.2. Extension Identification

In NETCONF capabilities are advertised in messages exchanged during session establishment. If multiple capability are used then the each define their own xml namespace.

A.2.6.1.3. Namespace Collision Protection

Again, in NETCONF capabilities are advertised in messages exchanged during session establishment. If multiple capability are used then the each define their own xml namespace.

Appendix B. NSCP Capabilities

This section defines the NSCP capabilities using the template in Appendix C of [RFC4741]. NSCP is broken in to several capabilities to allow for maximum flexibility

B.1. The NSCP Base Capability

B.1.1. Capability Name

NSCP Base

B.1.2. Overview

Exchange of this capability at session set up time indicates that the client and server both understand the base NSCP data model described in Section 2.1

B.1.3. Dependencies

None.

B.1.4. Capability Identifier

urn:ietf:dnsop:nscp:1.0

B.1.5. New Operations

None.

B.2. The NSCP Basic Control Capability

B.2.1. Capability Name

NSCP Basic Control

B.2.2. Overview

Exchange of this capability at session set up time indicates that the client and server both understand the basic NSCP control operations described in Section 2.2

There is no reconfig operation. It is assumed that this will happen automatically whenever the configuration is updated or that a form of commit-confirmed capability such as the one in [RFC4741] (section 8.4) will be used.

B.2.3. Dependencies

NSCP Base

B.2.4. Capability Identifier

urn:ietf:dnsop:nscp-control:1.0

B.2.5. New Operations

Note: Startup of a name server is a separate capability. This is to account for name servers that have a built in NSCP agent.

<stop>

Signals the name server to cleanly shutdown.

<reload>

Signals the name server reload a specified zone or all zones.

<restart>

Signals the name server to re-start.

B.3. The NSCP Start Control Capability

B.3.1. Capability Name

NSCP Start Control

B.3.2. Overview

Exchange of this capability at session set up time indicates that the client and server both understand the Start NSCP control operation described in Section 2.3

B.3.3. Dependencies

NSCP Start Control

B.3.4. Capability Identifier

urn:ietf:dnsop:nscp-start-control:1.0

B.3.5. New Operations

<start>

Signals the name server to start up.

Appendix C. YANG Data Model for Base NSCP capability

YANG Data Model for Base NSCP capability

```
module nscp {
  namespace "urn:ietf:dnsoop:nscp:1.0";
  prefix nscp;

  import yang-types { prefix yang; }
  import inet-types { prefix inet; }
  import ieee-types { prefix ieee; }

  organization
    "IETF";
  description
    "Base data model for NSCP.";

  container server {
    description "root of configuration and data";
    container statistics {
      description "container to hold statistics elements";
      leaf rr {
        description "Count of responses received.";
        config false;
        type yang:counter64;
      }
      leaf rnxld {
        description "Count of NXDomain received";
        config false;
        type yang:counter64;
      }
      leaf rdup {
        description "Count of duplicate responses";
        config false;
        type yang:counter64;
      }
      leaf rfail {
        description "Count of SERVFAIL responses received";
        config false;
        type yang:counter64;
      }
      leaf rferr {
        description "Count of FORMERR responses received";
        config false;
        type yang:counter64;
      }
      leaf rerr {
        description "Count of other errors received";
```

```
        config false;
        type yang:counter64;
    }
    leaf raxfr {
        description "Count of AXFR initiated";
        config false;
        type yang:counter64;
    }
    leaf rlame {
        description "Count of lame delegations received";
        config false;
        type yang:counter64;
    }
    leaf ssysq {
        description "Count of NS address fetches";
        config false;
        type yang:counter64;
    }
    leaf sans {
        description "Count of answers sent";
        config false;
        type yang:counter64;
    }
    leaf sfwdq {
        description "Count of queries sent";
        config false;
        type yang:counter64;
    }
    leaf sdupq {
        description "Count of queries retried";
        config false;
        type yang:counter64;
    }
    leaf rq {
        description "Count of requests received";
        config false;
        type yang:counter64;
    }
    leaf snxd {
        description "Count of NXDomain sent";
        config false;
        type yang:counter64;
    }
    leaf ruq {
        description "Count of non-recursive queries rejected";
        config false;
        type yang:counter64;
    }
}
```

```

    leaf rurq {
        description "Count of recursive queries rejected";
        config false;
        type yang:counter64;
    }
    leaf ruxfr {
        description "Count of XFR rejected";
        config false;
        type yang:counter64;
    }
    leaf ruupd {
        description "Count of updates rejected";
        config false;
        type yang:counter64;
    }
}

container dnssec-policy {
    description "DNSSEC policy";
    leaf-list secure {
        description
            "List of domains that are expected to be secure";
        type inet:domain-name;
    }
    leaf-list nonsecure {
        description
            "List of domains that are expected to be insecure";
        type inet:domain-name;
    }
    leaf KASP-database {
        description
            "Path or URL (To be decided) to allow the policy
            database to be located.";
        type string;
    }
}

container peers {
    description
        "Container for peer elements. May be used for partial
        locking";
    list peer {
        key "name";
        leaf name {
            description
                "name - used to refer to this peer elsewhere.";
            type string;
        }
    }
}

```



```

    list address {
      description
        "Address and port that will identify this peer";
      key "ip port";
      leaf ip {
        type inet:ip-prefix;
      }
      leaf port {
        type inet:port-number;
      }
    }
  container key {
    description
      "key that will be used to identify this peer. A
       choice is used to allow augmentation in
       the future";
    choice type {
      case a {
        leaf hmac-md5 {
          type string;
        }
      }
      case b {
        leaf hmac-sha1 {
          type string;
        }
      }
    }
  }
}

container panorama {
  description "A collection of views.";
  list view {
    key "name";
    leaf name {
      type string;
    }
  }
  list listen-on {
    description
      "Address and port that the nameserver will
       listen on.";
    key "ip port";
    leaf ip {
      type inet:ip-address;
    }
    leaf port {
      type inet:port-number;
    }
  }
}

```

```

    }
  }
  leaf recursion {
    description "Is recursion enabled for this view?";
    type enumeration {
      enum on;
      enum off;
    }
  }
  leaf validation {
    description "Is validation enabled in this view?";
    type enumeration {
      enum on;
      enum off;
    }
  }
  list zone {
    description "a zone";
    key "zone-name";
    leaf zone-name {
      description "the name of the zone.";
      type string;
    }
    leaf-list master {
      description
        "A reference to a peer that will act as a
        master for this zone";
      type keyref {
        path "/server/peers/peer/name";
      }
    }
    leaf-list slave {
      description
        "A reference to a peer that will act as a
        slave for this zone";
      type keyref {
        path "/server/peers/peer/name";
      }
    }
    leaf-list notify {
      description
        "A reference to a peer that notifies should
        be sent to for this zone";
      type keyref {
        path "/server/peers/peer/name";
      }
    }
  }
}

```

```

    container acl {
      description "The acl for this view";
      list ace {
        description "An access control entry.";
        key "identifier";
        leaf identifier {
          description
            "The name of a peer that will get
             this access.";
          type keyref {
            path "/server/peers/peer/name";
          }
        }
      }
      leaf-list access {
        description
          "The type of access that this peer
           will receive.";
        type enumeration {
          enum none;
          enum notify;
          enum query;
          enum recursion;
          enum transfer;
          enum update;
        }
      }
    }
  }
}

```

Appendix D. YANG Data Model for the NSCP Basic Control Capability

```

module nscp-basic-control {
  namespace "urn:ietf:dnsop:nscp-basic-control:1.0";
  prefix nscp-basic-control;

  organization
    "IETF";

  description
    "Adds basic control to NSCP.";

  rpc server-stop {
    /* Will cause server to stop cleanly. No inputs */
    output {
      leaf status {
        /* Will return something like "server stopping" */
        type string;
      }
    }
  }
  rpc server-reload {
    /* Will cause server to reload the
    * running configuration. No inputs */
    output {
      leaf status {
        /* Will return something like "server reloading" */
        type string;
      }
    }
  }
  rpc server-restart {
    /* Will cause server to reload the
    * running configuration. No inputs */
    output {
      leaf status {
        /* Will return something like "server restarting" */
        type string;
      }
    }
  }
}

```

Appendix E. YANG Data Model for the NSCP Start Control Capability

```
module nscp-start-control {
  namespace "urn:ietf:dnsop:nscp-start-control:1.0";
  prefix nscp-start-control;

  import yang-types { prefix yang; }
  import inet-types { prefix inet; }
  import ieee-types { prefix ieee; }

  organization
    "IETF";

  description
    "Adds start control to the base data model for NSCP.";

  rpc server-start {
    /* Will cause server to stop cleanly. No inputs */
    output {
      leaf status {
        /* Will return something like "server starting" */
        type string;
      }
    }
  }
}
```

Authors' Addresses

John A Dickinson
Sinodun Internet Technologies
Stables 4 Suite 11
Howbery Park
Wallingford, Oxfordshire OX10 8BA
UK

Email: jad@sinodun.com

Stephen Morris
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063
US

Email: stephen@isc.org

Roy Arends
Nominet UK
Minerva House
Edmund Halley Road
Oxford Science Park
Oxford, Oxfordshire OX4 4DQ
UK

Email: roy.arends@nominet.org.uk

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 28, 2011

S. Morris
ISC
J. Ihren
Netnod
J. Dickinson
Sinodun
October 25, 2010

DNSSEC Key Timing Considerations
draft-ietf-dnsop-dnssec-key-timing-01.txt

Abstract

This document describes the issues surrounding the timing of events in the rolling of a key in a DNSSEC-secured zone. It presents timelines for the key rollover and explicitly identifies the relationships between the various parameters affecting the process.

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 <http://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 April 28, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|----|
| 1. Introduction | 3 |
| 1.1. Key Rolling Considerations | 3 |
| 1.2. Types of Keys | 4 |
| 1.3. Terminology | 4 |
| 2. Rollover Methods | 4 |
| 2.1. ZSK Rollovers | 4 |
| 2.2. KSK Rollovers | 6 |
| 2.3. Summary | 7 |
| 3. Key Rollover Timelines | 8 |
| 3.1. Key States | 8 |
| 3.2. Zone-Signing Key Timelines | 9 |
| 3.2.1. Pre-Publication Method | 9 |
| 3.2.2. Double-Signature Method | 13 |
| 3.2.3. Double-RRSIG Method | 14 |
| 3.3. Key-Signing Key Rollover Timelines | 17 |
| 3.3.1. Double-Signature Method | 17 |
| 3.3.2. Double-DS Method | 20 |
| 3.3.3. Double-RRset Method | 22 |
| 3.3.4. Interaction with Configured Trust Anchors | 25 |
| 3.3.4.1. Addition of KSK | 25 |
| 3.3.4.2. Removal of KSK | 25 |
| 3.3.5. Introduction of First KSK | 26 |
| 4. Standby Keys | 27 |
| 5. Algorithm Considerations | 28 |
| 6. Limitation of Scope | 28 |
| 7. Summary | 28 |
| 8. IANA Considerations | 29 |
| 9. Security Considerations | 29 |
| 10. Acknowledgements | 29 |
| 11. Change History | 29 |
| 12. References | 30 |
| 12.1. Normative References | 30 |
| 12.2. Informative References | 31 |
| Appendix A. List of Symbols | 31 |
| Authors' Addresses | 35 |

1. Introduction

1.1. Key Rolling Considerations

When a zone is secured with DNSSEC, the zone manager must be prepared to replace ("roll") the keys used in the signing process. The rolling of keys may be caused by compromise of one or more of the existing keys, or it may be due to a management policy that demands periodic key replacement for security or operational reasons. In order to implement a key rollover, the keys need to be introduced into and removed from the zone at the appropriate times. Considerations that must be taken into account are:

- o DNSKEY records and associated information (such as RRSIG records created with the key or the associated DS records) are not only held at the authoritative nameserver, they are also cached at client resolvers. The data on these systems can be interlinked, e.g. a validating resolver may try to validate a signature retrieved from a cache with a key obtained separately.
- o A query for the key RRset returns all DNSKEY records in the zone. As there is limited space in the UDP packet (even with EDNS0 support), dead key records must be periodically removed. (For the same reason, the number of stand-by keys in the zone should be restricted to the minimum required to support the key management policy.)
- o Zone "boot-strapping" events, where a zone is signed for the first time, can be common in configurations where a large number of zones are being served. Procedures should be able to cope with the introduction of keys into the zone for the first time as well as "steady-state", where the records are being replaced as part of normal zone maintenance.
- o To allow for an emergency re-signing of the zone as soon as possible after a key compromise has been detected, stand-by keys (additional keys over and above those used to sign the zone) need to be present.

Management policy, e.g. how long a key is used for, also needs to be considered. However, the point of key management logic is not to ensure that a "rollover" is completed at a certain time but rather to ensure that no changes are made to the state of keys published in the zone until it is "safe" to do so ("safe" in this context meaning that at no time during the rollover process does any part of the zone ever go bogus). In other words, although key management logic enforces policy, it may not enforce it strictly.

1.2. Types of Keys

Although DNSSEC validation treats all keys equally, [RFC4033] recognises the broad classification of zone-signing keys (ZSK) and key-signing keys (KSK). A ZSK is used to authenticate information within the zone; a KSK is used to authenticate the key set in the zone. The main implication for this distinction concerns the consistency of information during a rollover.

During operation, a validating resolver must use separate pieces of information to perform an authentication. At the time of authentication, each piece of information may be in the validating resolver's cache or may need to be retrieved from the authoritative server. The rollover process needs to happen in such a way that at all times through the rollover the information is consistent. With a ZSK, the information is the RRSIG (plus associated RRset) and the DNSKEY. These are both obtained from the same zone. In the case of the KSK, the information is the DNSKEY and DS RRset with the latter being obtained from a different zone.

There are similarities in the rolling of ZSKs and KSKs: replace the RRSIG (plus RR) by the DNSKEY and replace the DNSKEY by the DS, and the ZSK rolling algorithms are virtually the same as the KSK algorithms. However, there are a number of differences, and for this reason the two types of rollovers are described separately in this document.

1.3. Terminology

The terminology used in this document is as defined in [RFC4033] and [RFC5011].

A large number of symbols are used to identify times, intervals, etc. All are listed in Appendix A.

2. Rollover Methods

2.1. ZSK Rollovers

A ZSK can be rolled in one of three ways:

- o Pre-Publication. Described in [RFC4641], the new key is introduced into the DNSKEY RRset, leaving the existing keys and signatures in place. This state of affairs remains in place for long enough to ensure that any DNSKEY RRsets cached in client validating resolvers contain both keys. At that point signatures created with the old key can be replaced by those created with the

new key, and the old signatures removed. During the re-signing process (which may or may not be atomic depending on how the zone is managed), it doesn't matter which key an RRSIG record retrieved by a client was created with; clients with a cached copy of the DNSKEY RRset will have a copy containing both the old and new keys.

Once the zone contains only signatures created with the new key, there is an interval during which RRSIG records created with the old key expire from client caches. After this, there will be no signatures anywhere that were created using the old key, and it can be removed from the DNSKEY RRset.

- o Double-Signature. Also mentioned in [RFC4641], this involves introducing the new key into the zone and using it to create additional RRSIG records; the old key and existing RRSIG records are retained. During the period in which the zone is being signed (again, the signing process may not be atomic), client resolvers are always able to validate RRSIGs: any combination of old and new DNSKEY RRset and RRSIG allows at least one signature to be validated.

Once the signing process is complete and enough time has elapsed to allow all old information to expire from caches, the old key and signatures can be removed from the zone. As before, during this period any combination of DNSKEY RRset and RRSIG will allow validation of at least one signature.

- o Double-RRSIG. Strictly speaking, the use of the term "Double-Signature" above is a misnomer as the method is not only double signature, it is also double key as well. A true Double-Signature method (here called the Double-RRSIG method) involves introducing new signatures in the zone (while still retaining the old ones) but not the new key.

Once the signing process is complete and enough time has elapsed to ensure that all caches that may contain an RR and associated RRSIG to have a copy of both signatures, the ZSK is changed. After a further interval during which the old DNSKEY RRset expires from caches, the old signatures are removed from the zone.

Of three methods, Double-Signature is the simplest conceptually - introduce the new key and new signatures, then approximately one TTL later remove the old key and signatures. The drawback of this method is a noticeable increase in the size of the DNSSEC data, affecting both the overall size of the zone and the size of the responses.

Pre-Publication is more complex - introduce the new key,

approximately one TTL later sign the records, and approximately one TTL after that remove the old key. However, the amount of DNSSEC data is kept to a minimum which reduces the impact on performance.

The Double-RRSIG combines the increase in data volume of the Double-Signature method with the complexity of Pre-Publication. It has few (if any) advantages and a description is only included here for completeness.

2.2. KSK Rollovers

In the ZSK case the issue for the validating resolver is to ensure that it has access to the ZSK that corresponds to a particular signature. In the KSK case this can never be a problem as the KSK is only used for one signature (that over the DNSKEY RRset) and both the key the signature travel together. Instead, the issue is to ensure that the KSK is trusted.

Trust in the KSK is either due to the existence of an explicitly configured trust anchor in the validating resolver or DS record in the parent zone (which is itself trusted). If the former, [RFC5011] timings will be needed to roll the keys. If the latter, the rollover algorithm will need to involve the parent zone in the addition and removal of DS records, so timings are not wholly under the control of the zone manager. (The zone manager may elect to include [RFC5011] timings in the key rolling process so as to cope with the possibility that the key has also been explicitly configured as a trust anchor.)

It is important to note that this does not preclude the development of key rollover logic; in accordance with the goal of the rollover logic being able to determine when a state change is "safe", the only effect of being dependent on the parent is that there may be a period of waiting for the parent to respond in addition to any delay the key rollover logic requires. Although this introduces additional delays, even with a parent that is less than ideally responsive the only effect will be a slowdown in the rollover state transitions. This may cause a policy violation, but will not cause any operational problems.

Like the ZSK case, there are three methods for rolling a KSK:

- o Double-Signature: Also known as Double-DNSKEY, the new KSK is added to the DNSKEY RRset which is then signed with both the old and new key. After waiting for the old RRset to expire from caches, the DS record in the parent zone is changed. After waiting a further interval for this change to be reflected in caches, the old key is removed from the RRset. (The name "Double-Signature" is used because, like the ZSK method of the same name,

the new key is introduced and immediately used for signing.)

- o Double-DS: the new DS record is published. After waiting for this change to propagate into the caches of all validating resolvers, the KSK is changed. After a further interval during which the old DNSKEY RRset expires from caches, the old DS record is removed.
- o Double-RRset: the new KSK is added to the DNSKEY RRset which is then signed with both the old and new key, and the new DS record added to the parent zone. After waiting a suitable interval for the old DS and DNSKEY RRsets to expire from validating resolver caches, the old DNSKEY and DS record are removed.

In essence, "Double-Signature" means that the new KSK is introduced first and used to sign the DNSKEY RRset. The DS record is changed, and finally the old KSK removed. With "Double-DS" it is the other way around. Finally, Double-RRset does both updates more or less in parallel.

The strategies have different advantages and disadvantages:

- o Double-Signature minimizes the number of interactions with the parent zone. However, for the period of the rollover the DNSKEY RRset is signed with two KSKs, so increasing the size of the response to a query for this data.
- o Double-DS keeps the size of the DNSKEY RRset to a minimum, but does require the additional administrative overhead of two interactions with the parent to roll a KSK. (Although this can be mitigated if the parent has the ability for a child zone to schedule the withdrawal of the old key at the same time as the introduction of the new one.)
- o Finally, Double-RRset allows the rollover to be done in roughly half the time of the other two methods; it also supports policies that require a period of running with old and new KSKs simultaneously. However, it does have the disadvantages of both the other two methods - it requires two signatures during the period of the rollover and two interactions with the parent.

2.3. Summary

The methods can be summarised as follows:

| ZSK Method | KSK Method | Description |
|------------------|------------------|--|
| Pre-Publication | (not applicable) | Publish the DNSKEY before the RRSIG. |
| Double-Signature | Double-Signature | Publish the DNSKEY and RRSIG at same time. (For a KSK, this happens before the DS is published.) |
| Double-RRSIG | (not applicable) | Publish RRSIG before the DNSKEY. |
| (not applicable) | Double-DS | Publish DS before the DNSKEY. |
| (not applicable) | Double-RRset | Publish DNSKEY and DS in parallel. |

Table 1

3. Key Rollover Timelines

3.1. Key States

During the rolling process, a key moves through different states. These states are:

- Generated** The key has been created, but has not yet been used for anything.
- Published** The DNSKEY record - or information associated with it - is published in the zone, but predecessors of the key (or associated information) may be held in resolver caches.
- The idea of "associated information" is used in rollover methods where RRSIG or DS records are published first and the DNSKEY is changed in an atomic operation. It allows the rollover still to be thought of as moving through a set of states. In the rest of this section, the term "key" should be taken to mean "key or associated information".
- Ready** The key has been published for long enough to guarantee that all caches that might contain a copy of the key RRset have a copy that includes this key.

- Active The key is in the zone and has started to be used to sign RRsets or authenticate the DNSKEY RRset. Note that when this state is entered, it might not be possible for every validating resolver to use the key for validation: the zone signing may not have finished, or the data might not have reached the resolver because of propagation delays and/or caching issues. If this is the case, the resolver will have to rely on the key's predecessor instead.
- Retired The key is in the zone but a successor key has become active. As there may still be information in caches that that require use of the key, it is being retained until this information expires.
- Dead The key is published in the zone but there is no information anywhere that requires its presence.
- Removed The key has been removed from the zone.
- There is one additional state, used where [RFC5011] considerations are in effect (see Section 3.3.4):
- Revoked The key is published for a period with the "revoke" bit set as a way of notifying validating resolvers that have configured it as a trust anchor that it is about to be removed from the zone.

3.2. Zone-Signing Key Timelines

3.2.1. Pre-Publication Method

The following diagram shows the time line of a particular ZSK and its replacement by its successor using the Pre-Publication method. Time increases along the horizontal scale from left to right and the vertical lines indicate events in the life of the key. The events are numbered; significant times and time intervals are marked.

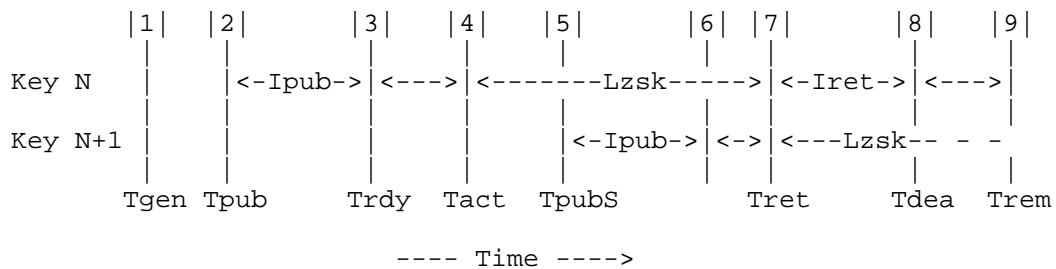


Figure 1: Timeline for a Pre-Publication ZSK rollover.

Event 1: key N is generated at the generate time (Tgen). Although there is no reason why the key cannot be generated immediately prior to publication in the zone (Event 2), some implementations may find it convenient to create a pool of keys in one operation and draw from that pool as required. For this reason, it is shown as a separate event. Keys that are available for use but not published are said to be generated.

Event 2: key N's DNSKEY record is put into the zone, i.e. it is added to the DNSKEY RRset which is then re-signed with the current key-signing key. The time at which this occurs is the key's publication time (Tpub), and the key is now said to be published. Note that the key is not yet used to sign records.

Event 3: before it can be used, the key must be published for long enough to guarantee that any resolver that has a copy of the DNSKEY RRset from the zone in its cache will have a copy of the RRset that includes this key: in other words, that any prior cached information about the DNSKEY RRset has expired.

This interval is the publication interval (Ipub) and, for the second or subsequent keys in the zone, is given by:

$$I_{pub} = D_{prp} + TTL_{key}$$

Here, D_{prp} is the propagation delay - the time taken for any change introduced at the master to replicate to all slave servers - which depends on the depth of the master-slave hierarchy. TTL_{key} is the time-to-live (TTL) for the DNSKEY records in the zone. The sum is therefore the time taken for existing DNSKEY records to expire from the caches of downstream validating resolvers, regardless of the nameserver from which they were retrieved.

In the case of the first key in the zone, Ipub is slightly different because it is not information about a DNSKEY RRset that may be

cached, it is information about its absence. In this case:

$$I_{pub} = D_{prp} + I_{ngc}$$

where I_{ngc} is the negative cache interval from the zone's SOA record, calculated according to [RFC2308] as the minimum of the TTL of the SOA record itself (TTL_{soa}), and the "minimum" field in the record's parameters (SOA_{min}), i.e.

$$I_{ngc} = \min(TTL_{soa}, SOA_{min})$$

After a delay of I_{pub} , the key is said to be ready and could be used to sign records. The time at which this event occurs is the key's ready time ($Trdy$), which is given by:

$$Trdy = T_{pub} + I_{pub}$$

Event 4: at some later time, the key starts being used to sign RRsets. This point is the active time ($Tact$) and after this, the key is said to be active.

Event 5: while this key is active, thought must be given to its successor (key $N+1$). As with the introduction of the currently active key into the zone, the successor key will need to be published at least I_{pub} before it is used. Denoting the publication time of the successor key by T_{pubS} , then:

$$T_{pubS} \leq Tact + L_{zsk} - I_{pub}$$

Here, L_{zsk} is the length of time for which a ZSK will be used (the ZSK lifetime). It should be noted that unlike the publication interval, L_{zsk} is not determined by timing logic, but by key management policy. L_{zsk} will be set by the operator according to their assessment of the risks posed by continuing to use a key and the risks associated with key rollover. However, operational considerations may mean a key is active for slightly more or less than L_{zsk} .

Event 6: while the key N is still active, its successor becomes ready. From this time onwards, it could be used to sign the zone.

Event 7: at some point the decision is made to start signing the zone using the successor key. This will be when the current key has been in use for an interval equal to the ZSK lifetime, hence:

$$T_{ret} = Tact + L_{zsk}$$

This point in time is the retire time (T_{ret}) of key N ; after this the

key is said to be retired. (This time is also the point at which the successor key becomes active.)

Event 8: the retired key needs to be retained in the zone whilst any RRSIG records created using this key are still published in the zone or held in resolver caches. (It is possible that a resolver could have an unexpired RRSIG record and an expired DNSKEY RRset in the cache when it is asked to provide both to a client. In this case the DNSKEY RRset would need to be looked up again.) This means that once the key is no longer used to sign records, it should be retained in the zone for at least the retire interval (Iret) given by:

$$\text{Iret} = \text{Dsgn} + \text{Dprp} + \text{TTLsig}$$

Dsgn is the delay needed to ensure that all existing RRsets have been re-signed with the new key. Dprp is (as described above) the propagation delay, required to guarantee that the updated zone information has reached all slave servers, and TTLsig is the TTL of the RRSIG records.

(It should be noted that an upper limit on the retire interval is given by:

$$\text{Iret} = \text{Lsig} + \text{Dskw}$$

where Lsig is the lifetime of a signature (i.e. the interval between the time the signature was created and the signature end time), and Dskw is the clock skew - the maximum difference in time between the server and a validating resolver. The reasoning here is that whatever happens, a key only has to be available while any signatures created with it are valid. Wherever a signature record is held - published in the zone and/or held in a resolver cache - it won't be valid for longer than Lsig after it was created. The Dskw term is present to account for the fact that the signature end time is an absolute time rather than interval, and systems may not agree exactly about the time.)

The time at which all RRSIG records created with this key have expired from resolver caches is the dead time (Tdea), given by:

$$\text{Tdea} = \text{Tret} + \text{Iret}$$

...at which point the key is said to be dead.

Event 9: at any time after the key becomes dead, it can be removed from the zone and the DNSKEY RRset re-signed with the current key-signing key. This time is the removal time (Trem), given by:

$Trem \geq Tdea$

...at which time the key is said to be removed.

3.2.2. Double-Signature Method

In the Double-Signature method, both the new key and signatures created using it are introduced at the same time. After a period during which this information propagates to validating resolver caches, the old key and signature are removed. The time line for the method is shown below:

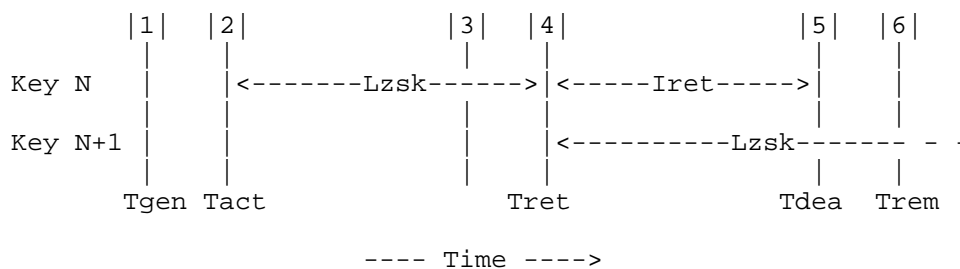


Figure 2: Timeline for a Double-Signature ZSK rollover.

Event 1: key N is generated at the generate time (Tgen). Although there is no reason why the key cannot be generated immediately prior to publication in the zone (Event 2), some implementations may find it convenient to create a pool of keys in one operation and draw from that pool as required. For this reason, it is shown as a separate event. Keys that are available for use but not published are said to be generated.

Event 2: key N is added to the DNSKEY RRset and is immediately used to sign the zone; existing signatures in the zone are not removed. This is the active time (Tact) and the key is said to be active.

Event 3: at some time later, the predecessor key (key N-1) and its signatures can be withdrawn from the zone. (The timing of key removal is discussed further in the description of event 5.)

Event 4: the successor key (key N+1) is introduced into the zone and starts being used to sign RRsets. The successor is key is now active and the current key (key N) is said to be retired. This time is the retire time of the key (Tret); it is also the active time of the successor key (TactS).

$$T_{ret} = T_{act} + L_{zsk}$$

Event 5: before key N can be withdrawn from the zone, all RRsets that need to be signed must have been signed by the successor key (as a result, all these RRsets are now signed twice, once by key N and once by its successor) and the information must have reached all validating resolvers that may have RRsets from this zone cached.

This takes I_{ret} , the retire interval, given by the expression:

$$I_{ret} = D_{sgn} + D_{prp} + \max(TTL_{key}, TTL_{sig})$$

As before, D_{sgn} is the time taken to sign the zone with the new key and D_{prp} is the propagation delay. The final term is the period to wait for old key and signature data to expire from caches. After the end of this interval, key N is said to be dead. This occurs at the dead time (T_{dea}) so:

$$T_{dea} = T_{ret} + I_{ret}$$

Event 6: at some later time key N and its signatures can be removed from the zone. This is the removal time T_{rem} , given by:

$$T_{rem} \geq T_{dea}$$

3.2.3. Double-RRSIG Method

As noted above, "Double-Signature" is simultaneous rollover of both signature and key. The time line for a pure Double-Signature ZSK rollover (the Double-RRSIG method) - where new signatures are introduced, the key changed, and finally the old signatures removed - is shown below:

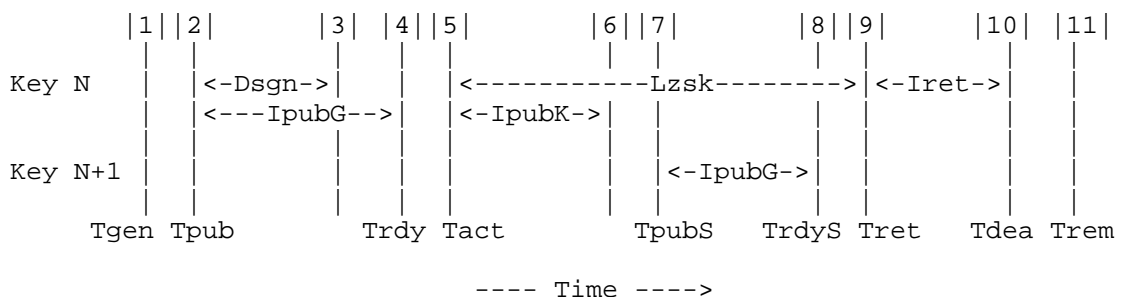


Figure 3: Timeline for a Double-Signature ZSK rollover.

Event 1: key N is generated at the generate time (Tgen). Although there is no reason why the key cannot be generated immediately prior to publication in the zone (Event 2), some implementations may find it convenient to create a pool of keys in one operation and draw from that pool as required. For this reason, it is shown as a separate event. Keys that are available for use but not published are said to be generated.

Event 2: key N is used to sign the zone but existing signatures are retained. Although the new ZSK is not published in the zone at this point, for analogy with the other ZSK rollover methods and because this is the first time that key information is visible (albeit indirectly through the created signatures) this time is called the publish time (Tpub).

Event 3: after the signing interval, Dsgn, all RRsets that need to be signed have been signed by the new key. (As a result, all these RRsets are now signed twice, once by the existing key and once by the (still-absent) key N.

Event 4: there is now a delay while the this information reaches all validating resolvers that may have RRsets from the zone cached. This interval is given by the expression:

$$Dprp + TTLsig$$

...comprising the delay for the information to propagate through the nameserver infrastructure plus the time taken for signature information to expire from caches.

Again in analogy with other key rollover methods, this is defined as key N's ready time (Trdy) and the key is said to be in the ready state. (Although the key is not in the zone, it is ready to be used.) The interval between the publication and ready times is the publication interval of the signature, IpubG, i.e.

$$Trdy = Tpub + IpubG$$

where

$$IpubG = Dsgn + Dprp + TTLsig$$

Event 5: at some later time the predecessor key is removed and the key N added to the DNSKEY RRset. As all the RRs have signatures created by the old and new keys, the records can still be authenticated. This time is the active time (Tact) and the key is now said to be active.

Event 6: After IpubK - the publication interval of the key - the newly added DNSKEY RRset has propagated through to all validating resolvers. At this point the old signatures can be removed from the zone. IpubK is given by:

$$\text{IpubK} = \text{Dprp} + \text{TTLkey}$$

Event 7: as before, at some later time thought must be given to rolling the key. The first step is to publish signatures created by the successor key (key N+1) early enough so that key N can be replaced after it has been active for its scheduled lifetime. This occurs at TpubS (the publication time of the successor), given by:

$$\text{TpubS} \leq \text{Tact} + \text{Lzsk} - \text{IpubG}$$

Event 8: the signatures have propagated through the zone and the new key could be added to the zone. This time is the ready time of the successor (TrdyS).

$$\text{TrdyS} = \text{TpubS} + \text{IpubG}$$

... where IpubG is as defined above.

Event 9: at some later time key N is removed from the zone and the successor key added. This is the retire time of the key (Tret).

Event 10: The signatures must remain in the zone for long enough that the new DNSKEY RRset has had enough time to propagate to all caches. Once caches contain the new DNSKEY, the old signatures are no longer of use and can be considered to be dead. The time at which this occurs is the dead time (Tdea), given by:

$$\text{Tdea} = \text{Tret} + \text{Iret}$$

... where Iret is the retire interval, given by:

$$\text{Iret} = \text{IpubK}$$

Event 11: At some later time the signatures can be removed from the zone. Although the key has is not longer in the zone, this is information associated with it and so the time can be regarded as the key's remove time (Trem), given by:

$$\text{Trem} \geq \text{Tdea}$$

3.3. Key-Signing Key Rollover Timelines

3.3.1. Double-Signature Method

The Double-Signature method (also known as the double-DNSKEY method) involves introducing the new KSK to the zone and waiting until its presence has been registered by all validating resolvers. At this point, the DS record in the parent is changed. Once that change has propagated to all validating resolvers, the old KSK is removed.

The timing diagram for such a rollover is:

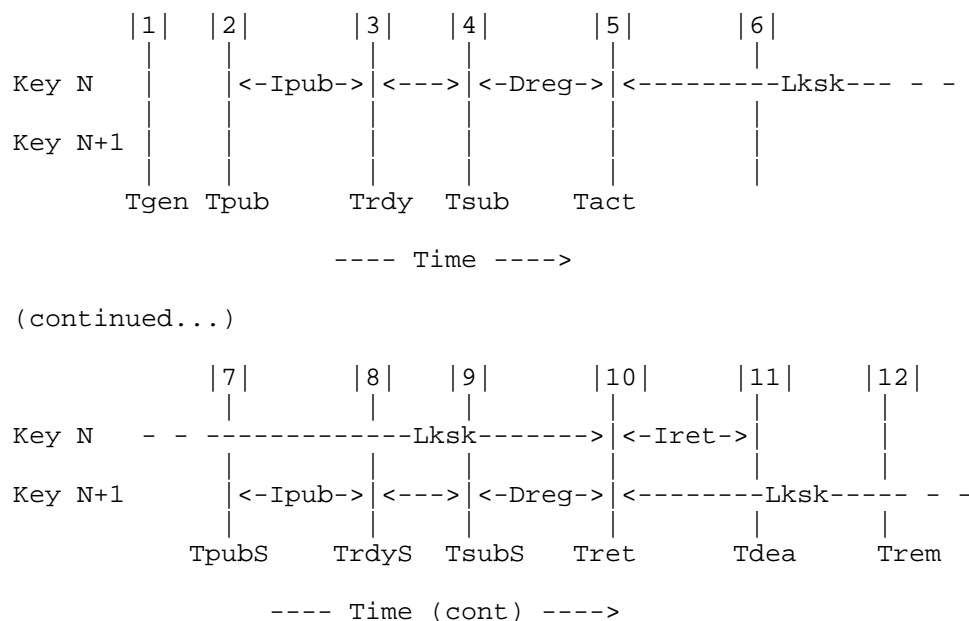


Figure 4: Timeline for a Double-Signature KSK rollover.

Event 1: key N is generated at time Tgen. As before, although there is no reason why the key cannot be generated immediately prior to publication, some implementations may find it convenient to create a central pool of keys and draw from it. For this reason, it is again shown as a separate event.

Event 2: key N is introduced into the zone; it is added to the DNSKEY RRset, which is then signed by key N and all currently active KSKs. (So at this point, the DNSKEY RRset is signed by both key N and its

predecessor KSK. If other KSKs were active, it is signed by these as well.) This is the publication time (T_{pub}); after this the key is said to be published.

Event 3: before it can be used, the key must be published for long enough to guarantee that any validating resolver that has a copy of the DNSKEY RRset from the zone in its cache will have a copy of the RRset that includes this key: in other words, that any prior cached information about the DNSKEY RRset has expired.

The interval is the publication interval (I_{pub}) and, for the second or subsequent KSKs in the zone, is given by:

$$I_{pub} = D_{prp} + TTL_{key}$$

... where D_{prp} is the propagation delay for the zone and TTL_{key} the TTL for the DNSKEY RRset. The time at which this occurs is the key's ready time, $Trdy$, given by:

$$Trdy = T_{pub} + I_{pub}$$

Event 4: at some later time, the DS RR corresponding to the new KSK is submitted to the parent zone for publication. This time is the submission time, T_{sub} .

Event 5: the DS record is published in the parent zone. As this is the point at which all information for authentication - both DNSKEY and DS record - is available in the two zones, it is the active time of the key:

$$T_{act} = T_{sub} + D_{reg}$$

... where D_{reg} is the registration delay, the time taken after the DS record has been received by the parent zone manager for it to be placed in the zone. (Parent zones are often managed by different entities, and this term accounts of the organisational overhead of transferring a record.)

Event 6: at some time later, all validating resolvers that have the DS RRset cached will have a copy that includes the new DS record. For the second or subsequent DS records, this interval is given by the expression:

$$D_{prpP} + TTL_{ds}$$

... where D_{prpP} is the propagation delay in the parent zone and TTL_{ds} the TTL assigned to DS records in that zone.

In the case of the first DS record for the zone in question, the expression is slightly different because it is not information about a DS RRset that may be cached, it is information about its absence. In this case, the interval is:

$$\text{DprpP} + \text{IngcP}$$

where IngcP is the negative cache interval from the zone's SOA record, calculated according to [RFC2308] as the minimum of the TTL of the parent SOA record itself (TTLsoaP), and the "minimum" field in the record's parameters (SOAminP), i.e.

$$\text{IngcP} = \min(\text{TTLsoaP}, \text{SOAminP})$$

Event 7: while key N is active, thought needs to be given to its successor (key N+1). At some time before the scheduled end of the KSK lifetime, the successor KSK is introduced into the zone and is used to sign the DNSKEY RRset. (As before, this means that the DNSKEY RRset is signed by both the current and successor KSK.) This is the publication time of the successor key, TpubS .

Event 8: after an interval Ipub , the successor key becomes ready (in that all validating resolvers that have a copy of the DNSKEY RRset have a copy of this key). This is the successor ready time, TrdyS .

Event 9: at the successor submission time (TsubS), the DS record corresponding to the successor key is submitted to the parent zone.

Event 10: the successor DS record is published in the parent zone and the current DS record withdrawn. The current key is said to be retired and the time at which this occurs is Tret , given by:

The relationships between these times are:

$$\text{TpubS} \leq \text{Tact} + \text{Lsk} - \text{Dreg} - \text{Ipub}$$

$$\text{Tret} = \text{Tact} + \text{Lsk}$$

... where Lsk is the scheduled lifetime of the KSK.

Event 11: key N must remain in the zone until any validators that have the DS RRset cached have a copy of the DS RRset containing the new DS record. This interval is the retire interval, given by:

$$\text{Iret} = \text{DprpP} + \text{TTLds}$$

... where DprpP is the propagation delay in the parent zone and TTLds the TTL of a DS record.

As the key is no longer used for anything, it can also be said to be dead, in which case:

$$T_{dea} = T_{ret} + I_{ret}$$

Event 12: at some later time, key N is removed from the zone (at the remove time T_{rem}); the key is now said to be removed.

$$T_{rem} \geq T_{dea}$$

3.3.2. Double-DS Method

The Double-DS method is the reverse of the Double-Signature method is that it is the DS record that is pre-published (in the parent), and not the DNSKEY.

The timeline for the key rollover is shown below:

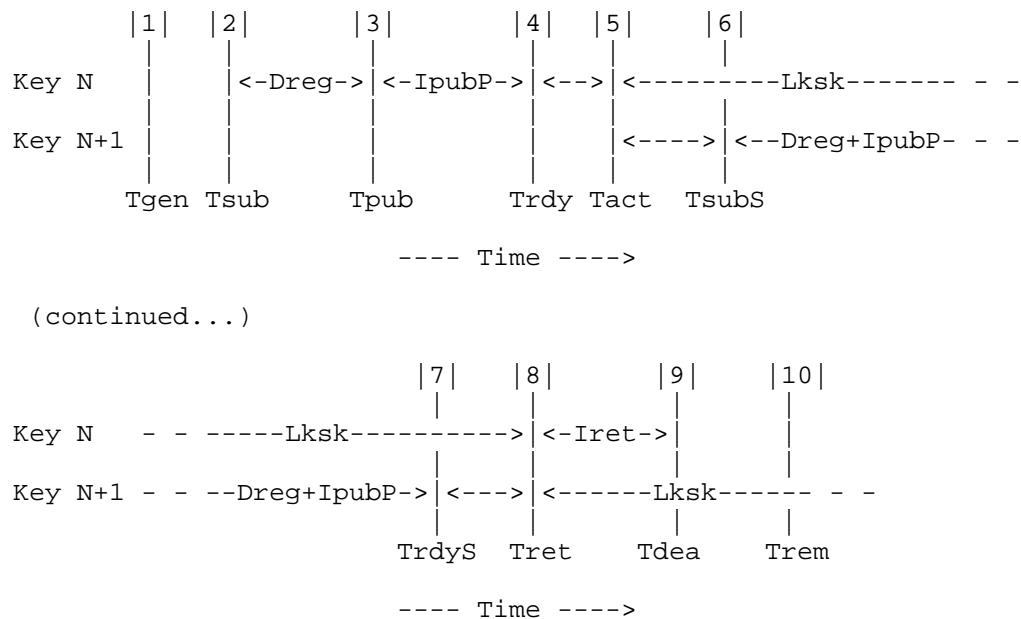


Figure 5: Timeline for a Double-DS KSK rollover.

Event 1: key N is generated at time T_{gen} . As before, although there is no reason why the key cannot be generated immediately prior to publication, some implementations may find it convenient to create a

central pool of keys and draw from it. For this reason, it is again shown as a separate event.

Event 2: the DS record corresponding to key N is submitted for publication in the parent zone. This time is the submission time (T_{sub}).

Event 3: after the registration delay, D_{reg} , the DS record is published in the parent zone. This is the publication time T_{pub} , given by:

$$T_{pub} = T_{sub} + D_{reg}$$

Event 4: at some later time, any validating resolver that has copies of the DS RRset in its cache will have a copy of the DS record for key N. At this point, key N, if introduced into the DNSKEY RRset, could be used to validate the zone. For this reason, this time is known as the key's ready time, T_{rdy} , and is given by:

$$T_{rdy} = T_{pub} + I_{pubP}$$

I_{pubP} is the parent publication interval and is given by the expression:

$$I_{pubP} = D_{prpP} + TTL_{ds}$$

... where D_{prpP} is the propagation delay in the parent zone and TTL_{ds} the TTL assigned to DS records in that zone.

Event 5: at some later time, the key rollover takes place. The predecessor key is withdrawn from the DNSKEY RRset and the new key (key N) introduced and used to sign the RRset.

As both DS records have been in the parent zone long enough to ensure that they are in the cache of any validating resolvers that have the DS RRset cached, the zone can be authenticated throughout the rollover - either the resolver has a copy of the DNSKEY RRset (and associated RRSIGs) authenticated by the predecessor key, or it has a copy of the updated RRset authenticated with the new key.

This time is the key's active time (T_{act}) and at this point the key is said to be active.

Event 6: at some point thought must be given to key replacement. The DS record for the successor key must be submitted to the parent zone at a time such that when the current key is withdrawn, any validating resolver that has DS records in its cache will have data about the DS record of the successor key. The time at which this occurs is the

submission time of the successor, given by:

$$T_{\text{subS}} \leq T_{\text{act}} + L_{\text{sk}} - I_{\text{pubP}} - D_{\text{reg}}$$

... where L_{sk} is the lifetime of the KSK.

Event 7: the successor key (key $N+1$) enters the ready state i.e. its DS record is now in the caches of all validating resolvers that have the parent DS RRset cached. (This is the ready time of the successor, T_{rdyS} .)

Event 8: when the current key has been active for its lifetime (L_{sk}), the current key is removed from the DNSKEY RRset and the successor key added; the RRset is then signed with the successor key. This point is the retire time of the key, T_{ret} , given by:

$$T_{\text{ret}} = T_{\text{act}} + L_{\text{sk}}$$

Event 9: at some later time, all copies of the old DNSKEY RRset have expired from caches and the old DS record is no longer needed. This is called the dead time, T_{dea} , and is given by:

$$T_{\text{dea}} = T_{\text{ret}} + I_{\text{ret}}$$

... where I_{ret} is the retire interval, given by:

$$I_{\text{ret}} = D_{\text{prp}} + T_{\text{TTLkey}}$$

As before, this term includes the time taken to propagate the RRset change through the master-slave hierarchy and the time take for the DNSKEY RRset to expire from caches.

Event 10: at some later time, the DS record is removed from the parent zone. This is the removal time (T_{rem}), given by:

$$T_{\text{rem}} \geq T_{\text{dea}}$$

3.3.3. Double-RRset Method

In the Double-RRset method, both the DS and DNSKEY records are changed at the same time, so for a period the zone can be authenticated with either key. The advantage of this method is its applicability in cases where zone management policy requires overlap of authentication keys during a roll.

The timeline for this rollover is shown below:

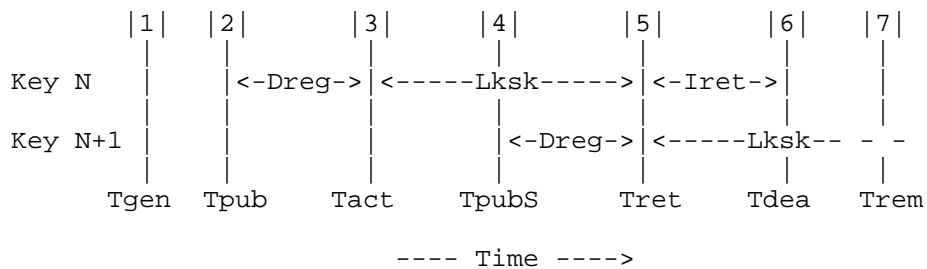


Figure 6: Timeline for a Double-RRset KSK rollover.

Event 1: key N is created at time Tgen and thereby immediately becomes generated. As before, although there is no reason why the key cannot be generated immediately prior to publication, some implementations may find it convenient to create a central pool of keys and draw from it. For this reason, it is again shown as a separate event.

Event 2: the key is added to and used for signing the DNSKEY RRset and is thereby published in the zone. At the same time the corresponding DS record is submitted to the parent zone for publication. This time is the publish time (Tpub) and the key is now said to be published.

Event 3: after Dreg, the registration delay, the DS record is published in the parent zone. At this point, the zones have all the information needed for a validating resolver to authenticate the zone, although the information may not yet have reached all validating resolver caches. This time is the active time (Tact) and the key is said to be active.

$$Tact = Tpub + Dreg$$

Event 4: at some point we need to give thought to key replacement. The successor key must be introduced into the zone (and its DS record submitted to the parent) at a time such that it becomes active when the current key has been active for its lifetime, Lksk. This time is TpubS, the publication time of the successor key, and is given by:

$$TpubS \leq Tact + Lksk - Dreg$$

... where Lksk is the lifetime of the KSK.

Event 5: the successor key's DS record appears in the parent zone and the successor key becomes active. At this point, the current key becomes retired. This occurs at Tret, given by:

$$Tret = Tact + Lksk$$

Event 6: the current DNSKEY and DS record must be retained in the zones until any validating resolver that has cached the DNSKEY and/or DS RRsets will have a copy of the data for the successor key. At this point the current key information is dead, as any validating resolver can perform authentication using the successor key. This is the dead time, $Tdea$, given by:

$$Tdea = Tret + Iret$$

... where $Iret$ is the retire interval. This depends on how long both the successor DNSKEY and DS records take to propagate through the nameserver infrastructure and thence into validator caches. These delays are the publication intervals of the child and parent zones respectively, so a suitable expression for $Iret$ is:

$$Iret = \max(IpubP, IpubC)$$

$IpubC$ is the publication interval of the DNSKEY in the child zone, $IpubP$ that of the DS record in the parent.

The child term comprises two parts - the time taken for the introduction of the DNSKEY record to be propagated to the downstream secondary servers (= $DprpC$, the child propagation delay) and the time taken for information about the DNSKEY RRset to expire from the validating resolver cache, i.e.

$$IpubC = DprpC + TTLkey$$

$TTLkey$ is the TTL for a DNSKEY record in the child zone. The parent term is similar:

$$IpubP = DprpP + TTLds$$

$DprpP$ the propagation delay in the parent zone and $TTLds$ the TTL for a DS record in the parent zone.

Event 7: at some later time, the DNSKEY record can be removed from the child zone and a request can be made to remove the DS record from the parent zone. This is the removal time, $Trem$ and is given by:

$$Trem \geq Tdea$$

3.3.4. Interaction with Configured Trust Anchors

Although the preceding sections have been concerned with rolling KSKs where the trust anchor is a DS record in the parent zone, zone managers may want to take account of the possibility that some validating resolvers may have configured trust anchors directly.

Rolling a configured trust anchor is dealt with in [RFC5011]. It requires introducing the KSK to be used as the trust anchor into the zone for a period of time before use, and retaining it (with the "revoke" bit set) for some time after use. The Double-Signature and Double-RRset methods can be adapted to include [RFC5011] recommendations so that the rollover will also be signalled to validating resolvers with configured trust anchors. (The recommendations are not suitable for the Double-DS method. Introducing the new key early and retaining the old key after use effectively converts it into a form of Double-RRset.)

3.3.4.1. Addition of KSK

When the new key is introduced, the publication interval (I_{pub}) in the Double-Signature method should also be subject to the condition:

$$I_{pub} \geq \max(30 \text{ days}, TTL_{key})$$

... where the right hand side of the expression is the add hold-down time defined in section 2.4.1 of [RFC5011].

In the Double-RRSIG method, the key should not be regarded as being active until the add hold-down time has passed. In other words, the following condition should be enforced:

$$T_{act} \geq T_{pub} + \max(30 \text{ days}, TTL_{key})$$

(Effectively, this means extending the lifetime of the key by an appropriate amount.)

3.3.4.2. Removal of KSK

The timeline for the removal of the key in both methods is modified by introducing a new state, "revoked". When the key reaches the end of the retire period, instead of being declared "dead", it is revoked; the "revoke" bit is set on the DNSKEY RR and is published in (and used to sign) the DNSKEY RRset. The key is maintained in this state for the "revoke" interval, I_{rev} , given by:

$I_{rev} \geq 30 \text{ days}$

... 30 days being the [RFC5011] remove hold-down time. After this time, the key is dead and can be removed from the zone when convenient.

3.3.5. Introduction of First KSK

There is an additional consideration when introducing a KSK into a zone for the first time, and that is that no validating resolver should be in a position where it can access the trust anchor before the KSK appears in the zone. To do so will cause the validating resolver to declare the zone to be bogus.

This is important: in the case of a secure parent, it means ensuring that the DS record is not published in the parent zone until there is no possibility that a validating resolver can obtain the record yet not be able to obtain the corresponding DNSKEY. In the case of an insecure parent, i.e. the initial creation of a new security apex, it is important to not configure trust anchors in validating resolvers until the DNSKEY RRset has had sufficient time to propagate. In both cases, this time is the trust anchor availability time (T_{taa}) given by:

$$T_{taa} \geq T_{pub} + I_{pubC}$$

where

$$I_{pubC} = D_{prpC} + TTL_{keyC}$$

or

$$I_{pubC} = D_{prpC} + IngC$$

The first expression applies if there was previously a DNSKEY RRset in the child zone, the expression for I_{pubC} including the TTL_{keyC} term to account for the time taken for that RRset to expire from caches. (It is possible that the zone was signed but that the trust anchor had not been submitted to the parent.)

If the introduction of the KSK caused the appearance of the first DNSKEY RRset in the child zone, the second expression applies in which the TTL_{keyC} term is replaced by $IngC$ to allow for the effect of negative caching.

As before, $IngC$ is the negative cache interval from the child zone's SOA record, calculated according to [RFC2308] as the minimum of the TTL of the SOA record itself (TTL_{soaC}), and the "minimum" field in

the record's parameters (SOAminC), i.e.

$$\text{IngcC} = \min(\text{TTLsoaC}, \text{SOAminC})$$

4. Standby Keys

Although keys will usually be rolled according to some regular schedule, there may be occasions when an emergency rollover is required, e.g. if the active key is suspected of being compromised. The aim of the emergency rollover is to allow the zone to be re-signed with a new key as soon as possible. As a key must be in the ready state to sign the zone, having at least one additional key (a standby key) in this state at all times will minimise delay.

In the case of a ZSK, a standby key only really makes sense with the Pre-Publication method. A permanent standby DNSKEY RR should be included in zone or successor keys could be introduced as soon as possible after a key becomes active. Either way results in an additional ZSK in the DNSKEY RRset that can immediately be used to sign the zone if the current key is compromised.

(Although in theory the mechanism could be used with both the Double-Signature and Double-RRSIG methods, it would require Pre-Publication of the signatures. Essentially, the standby key would be permanently active, as it would have to be periodically used to renew signatures. Zones would also permanently require two sets of signatures, something that could have a performance impact in large zones.)

A standby key can also be used with the Double-Signature and Double-DS methods of rolling a KSK. (The idea of a standby key in the Double-RRset effectively means having two active keys.) The Double-Signature method requires that the standby KSK be included in the DNSKEY RRset; rolling the key then requires just the introduction of the DS record in the parent. (Note that the DNSKEY should also be used to sign the DNSKEY RRset. As the RRset and its signatures travel together, merely adding the DNSKEY does not provide the desired time saving; to be used in a rollover requires that the DNSKEY RRset be signed with the standby key, and this introduces a delay whilst the RRset and its signatures propagate to the caches of validating resolvers. There is no time advantage over introducing a new DNSKEY and signing the RRset with it at the same time.)

In the Double-DS method of rolling a KSK, it is not a standby key that is present, it is a standby DS record in the parent zone. Whatever algorithm is used, the standby item of data can be introduced as a permanent standby, or be a successor introduced as early as possible.

5. Algorithm Considerations

The preceding sections have implicitly assumed that all keys and signatures are created using a single algorithm. However, [RFC4035] (section 2.4) states that "There MUST be an RRSIG for each RRset using at least one DNSKEY of each algorithm in the zone apex DNSKEY RRset".

Except in the case of an algorithm rollover - where the algorithms used to create the signatures are being changed - there is no relationship between the keys of different algorithms. This means that they can be rolled independently of one another. In other words, the key rollover logic described above should be run separately for each algorithm; the union of the results is included in the zone, which is signed using the active key for each algorithm.

6. Limitation of Scope

This document represents current thinking at the time of publication. However, the subject matter is evolving and it is more than likely that this document will need to be revised in the future.

Some of the techniques and ideas that DNSSEC operators are thinking about differ from those described in this document. Of note are alternatives to the strict split into KSK and ZSK key roles and the consequences for rollover logic from partial signing (i.e. when the new key initially only signs a fraction of the zone while leaving other signatures generated by the old key in place).

Furthermore, as noted in section 5, this document covers only rolling keys of the same algorithm, it does not cover transition to/from addition/deletion of different algorithm. Algorithm rollovers will require a separate document.

The reader is therefore reminded that DNSSEC is as of publication in early stages of deployment, and best practices will likely change in the near term.

7. Summary

For ZSKs, "Pre-Publication" is generally considered to be the preferred way of rolling keys. As shown in this document, the time taken to roll is wholly dependent on parameters under the control of the zone manager.

In contrast, "Double-RRset" is the most efficient method for KSK

rollover due to the ability to have new DS records and DNSKEY RRsets propagate in parallel. The time taken to roll KSKs may depend on factors related to the parent zone if the parent is signed. For zones that intend to comply with the recommendations of [RFC5011], in virtually all cases the rollover time will be determined by the RFC5011 "add hold-down" and "remove hold-down" times. It should be emphasized that this delay is a policy choice and not a function of timing values and that it also requires changes to the rollover process due to the need to manage revocation of trust anchors.

Finally, the treatment of emergency key rollover is significantly simplified by the introduction of stand-by keys as standard practice during all types of rollovers.

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

This document does not introduce any new security issues beyond those already discussed in [RFC4033], [RFC4034], [RFC4035] and [RFC5011].

10. Acknowledgements

The authors gratefully acknowledge help and contributions from Roy Arends and Wouter Wijngaards.

11. Change History

- o draft-ietf-dnsop-dnssec-key-timing-00
 - * Added section on limitation of scope.
- o draft-morris-dnsop-dnssec-key-timing-02
 - * General restructuring.
 - * Added descriptions of more rollovers (IETF-76 meeting).
 - * Improved description of key states and removed diagram.
 - * Provided simpler description of standby keys.
 - * Added section concerning first key in a zone.
 - * Moved [RFC5011] to a separate section.
 - * Various nits fixed (Alfred Hoenes, Jeremy Reed, Scott Rose, Sion Lloyd, Tony Finch).

- o draft-morris-dnsop-dnssec-key-timing-01
 - * Use latest boilerplate for IPR text.
 - * List different ways to roll a KSK (acknowledgements to Mark Andrews).
 - * Restructure to concentrate on key timing, not management procedures.
 - * Change symbol notation (Diane Davidowicz and others).
 - * Added key state transition diagram (Diane Davidowicz).
 - * Corrected spelling, formatting, grammatical and style errors (Diane Davidowicz, Alfred Hoenes and Jinmei Tatuya).
 - * Added note that in the case of multiple algorithms, the signatures and rollovers for each algorithm can be considered as more or less independent (Alfred Hoenes).
 - * Take account of the fact that signing a zone is not atomic (Chris Thompson).
 - * Add section contrasting pre-publication rollover with double signature rollover (Matthijs Mekking).
 - * Retained distinction between first and subsequent keys in definition of initial publication interval (Matthijs Mekking).
- o draft-morris-dnsop-dnssec-key-timing-00
 - Initial draft.

12. References

12.1. Normative References

- [RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, March 1998.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.
- [RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.
- [RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.
- [RFC5011] StJohns, M., "Automated Updates of DNS Security (DNSSEC) Trust Anchors", RFC 5011, September 2007.

12.2. Informative References

- [RFC4641] Kolkman, O. and R. Gieben, "DNSSEC Operational Practices", RFC 4641, September 2006.

Appendix A. List of Symbols

The document defines a number of symbols, all of which are listed here. All are of the form:

All symbols used in the text are of the form:

<TYPE><id><INST>

where:

<TYPE> is an upper-case character indicating what type the symbol is. Defined types are:

| | |
|-----|--|
| D | delay: interval that is a feature of the process |
| I | interval between two events |
| L | lifetime: interval set by the zone manager |
| SOA | parameter related to SOA RR |
| T | a point in time |
| TTL | TTL of a record |

T and I are self-explanatory. D, and L are also time periods, but whereas I values are intervals between two events (even if the events are defined in terms of the interval, e.g. the dead time occurs "retire interval" after the retire time), D, and L are fixed intervals. An "L" interval (lifetime) is chosen by the zone manager and is a feature of policy. A "D" interval (delay) is a feature of the process, probably outside control of the zone manager. SOA and TTL are used just because they are common terms.

<id> is lower-case and defines what object or event the variable is related to, e.g.

| | |
|-----|----------------|
| act | active |
| ngc | negative cache |
| pub | publication |

Finally, <INST> is a capital letter that distinguishes between the same variable applying to different instances of an object and is one of:

| | |
|---|-----------|
| C | child |
| G | signature |
| K | key |
| P | parent |
| S | successor |

The list of variables used in the text is:

| | |
|-------|--|
| Dprp | Propagation delay. The amount of time for a change made at a master nameserver to propagate to all the slave nameservers. |
| DprpC | Propagation delay in the child zone. |
| DprpP | Propagation delay in the parent zone. |
| Dreg | Registration delay. As a parent zone is often managed by a different organisation to that managing the child zone, the delays associated with passing data between zones is captured by this term. |
| Dskw | Clock skew. The maximum difference in time between the signing system and the resolver. |
| Dsgn | Signing delay. After the introduction of a new ZSK, the amount of time taken for all the RRs in the zone to be signed with it. |
| Ingc | Negative cache interval. |
| IngcP | Negative cache interval of the child zone. |

| | |
|---------|--|
| IngcP | Negative cache interval of the parent zone. |
| Ipub | Publication interval. The amount of time that must elapse after the publication of a key before it can be considered to have entered the ready state. |
| IpubC | Publication interval in the child zone. |
| IpubG | Publication interval for the signature. |
| IpubK | Publication interval for the key. |
| IpubP | Publication interval in the parent zone. |
| Iret | Retire interval. The amount of time that must elapse after a key enters the retire state for any signatures created with it to be purged from validating resolver caches. |
| Irev | Revoke interval. The amount of time that a KSK must remain published with the revoke bit set to satisfy [RFC5011] considerations. |
| Lksk | Lifetime of a key-signing key. This is the intended amount of time for which this particular KSK is regarded as the active KSK. Depending on when the key is rolled-over, the actual lifetime may be longer or shorter than this. |
| Lzsk | Lifetime of a zone-signing key. This is the intended amount of time for which the ZSK is used to sign the zone. Depending on when the key is rolled-over, the actual lifetime may be longer or shorter than this. |
| Lsig | Lifetime of a signature: the difference in time between the signature's expiration time and the time at which the signature was created. (Note that this is not the difference between the signature's expiration and inception times: the latter is usually set a small amount of time before the signature is created to allow for clock skew between the signing system and the validating resolver.) |
| SOAmin | Value of the "minimum" field from an SOA record. |
| SOAminC | Value of the "minimum" field from an SOA record in the child zone. |

| | |
|---------|---|
| SOAminP | Value of the "minimum" field from an SOA record in the parent zone. |
| Tact | Active time of the key; the time at which the key is regarded as the principal key for the zone. |
| TactS | Active time of the successor key. |
| Tdea | Dead time of a key. Applicable only to ZSKs, this is the time at which any record signatures held in validating resolver caches are guaranteed to be created with the successor key. |
| Tgen | Generate time of a key. The time that a key is created. |
| Tpub | Publish time of a key. The time that a key appears in a zone for the first time. |
| TpubS | Publish time of the successor key. |
| Trem | Removal time of a key. The time at which a key is removed from the zone. |
| Tret | Retire time of a key. The time at which a successor key starts being used to sign the zone. |
| Trdy | Ready time of a key. The time at which it can be guaranteed that validating resolvers that have key information from this zone cached have a copy of this key in their cache. (In the case of KSKs, should the validating resolvers also have DS information from the parent zone cached, the cache must include information about the DS record corresponding to the key.) |
| TrdyS | Ready time of a successor key. |
| Tsub | Submit time - the time at which the DS record of a KSK is submitted to the parent. |
| TsubS | Submit time of the successor key. |
| TTLds | Time to live of a DS record (in the parent zone). |
| TTLkey | Time to live of a DNSKEY record. |

TTLkeyC Time to live of a DNSKEY record in the child zone.

TTLsoa Time to live of a SOA record.

TTLsoaC Time to live of a SOA record in the child zone.

TTLsoaP Time to live of a SOA record in the parent zone.

TTLsig Time to live of an RRSIG record.

Ttaa Trust anchor availability time. The time at which a trust anchor record can be made available when a KSK is first introduced into a zone.

Authors' Addresses

Stephen Morris
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063
USA

Phone: +1 650 423 1300
Email: stephen@isc.org

Johan Ihren
Netnod
Franzengatan 5
Stockholm, SE-112 51
Sweden

Phone: +46 8615 8573
Email: johani@autonomica.se

John Dickinson
Sinodun Internet Technologies Ltd
Stables 4 Suite 11, Howbery Park
Wallingford, Oxfordshire OX10 8BA
UK

Phone: +44 1491 818120
Email: jad@sinodun.com

DNSOP
Internet-Draft
Obsoletes: 2541 (if approved)
Intended status: Informational
Expires: April 24, 2011

O. Kolkman
NLnet Labs
October 21, 2010

DNSSEC Operational Practices, Version 2
draft-ietf-dnsop-rfc4641bis-05

Abstract

This document describes a set of practices for operating the DNS with security extensions (DNSSEC). The target audience is zone administrators deploying DNSSEC.

The document discusses operational aspects of using keys and signatures in the DNS. It discusses issues of key generation, key storage, signature generation, key rollover, and related policies.

[When approved] This document obsoletes RFC 4641 as it covers more operational ground and gives more up-to-date requirements with respect to key sizes and the DNSSEC operations.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on April 24, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

| | |
|--|----|
| 1. Introduction | 5 |
| 1.1. The Use of the Term 'key' | 6 |
| 1.2. Time Definitions | 6 |
| 2. Keeping the Chain of Trust Intact | 7 |
| 3. Keys Generation and Storage | 8 |
| 3.1. Operational Motivation for Zone Signing and Key Signing Keys | 8 |
| 3.2. Practical Consequences of KSK and ZSK Separation | 10 |
| 3.2.1. Rolling a KSK that is not a trust-anchor | 10 |
| 3.2.2. Rolling a KSK that is a trust-anchor | 11 |
| 3.2.3. The use of the SEP flag | 12 |
| 3.3. Key Effectivity Period | 12 |
| 3.4. Cryptographic Considerations | 13 |
| 3.4.1. Key Algorithm | 13 |
| 3.4.2. Key Sizes | 14 |
| 3.4.3. Private Key Storage | 15 |
| 3.4.4. Key Generation | 16 |
| 3.4.5. Differentiation for 'High-Level' Zones? | 16 |
| 4. Signature Generation, Key Rollover, and Related Policies | 17 |
| 4.1. Key Rollovers | 17 |
| 4.1.1. Zone Signing Key Rollovers | 17 |

| | | |
|----------|---|----|
| 4.1.1.1. | Pre-Publish Zone Signing Key Rollover | 17 |
| 4.1.1.2. | Double Signature Zone Signing Key Rollover | 20 |
| 4.1.1.3. | Pros and Cons of the Schemes | 22 |
| 4.1.2. | Key Signing Key Rollovers | 22 |
| 4.1.2.1. | Special Considerations for RFC5011 KSK rollover | 24 |
| 4.1.3. | Difference Between ZSK and KSK Rollovers | 24 |
| 4.1.4. | Rollover for a Single Type Signing Key rollover | 25 |
| 4.1.5. | Algorithm rollovers | 27 |
| 4.1.5.1. | NSEC to NSEC3 algorithm rollover | 29 |
| 4.1.5.2. | Single Type Signing Scheme Algorithm Rollover | 30 |
| 4.1.5.3. | Algorithm rollover, RFC5011 style | 30 |
| 4.1.5.4. | Single Signing Type, RFC5011 style rollovers | 31 |
| 4.1.6. | Considerations for Automated Key Rollovers | 32 |
| 4.2. | Planning for Emergency Key Rollover | 33 |
| 4.2.1. | KSK Compromise | 33 |
| 4.2.1.1. | Keeping the Chain of Trust Intact | 34 |
| 4.2.1.2. | Breaking the Chain of Trust | 35 |
| 4.2.2. | ZSK Compromise | 35 |
| 4.2.3. | Compromises of Keys Anchored in Resolvers | 35 |
| 4.3. | Parent Policies | 36 |
| 4.3.1. | Initial Key Exchanges and Parental Policies Considerations | 36 |
| 4.3.2. | Storing Keys or Hashes? | 36 |
| 4.3.3. | Security Lameness | 37 |
| 4.3.4. | DS Signature Validity Period | 37 |
| 4.3.5. | Changing DNS Operators | 38 |
| 4.3.5.1. | Cooperationg DNS operators | 38 |
| 4.3.5.2. | Non Cooperationg DNS Operators | 40 |
| 4.4. | Time in DNSSEC | 41 |
| 4.4.1. | Time Considerations | 41 |
| 4.4.2. | Signature Validation Periods | 44 |
| 4.4.2.1. | Maximum Value | 44 |
| 4.4.2.2. | Minimum Value | 44 |
| 4.4.2.3. | Differentiation between RR sets | 45 |
| 4.4.2.4. | Other timing parameters in a zone | 46 |
| 5. | Next Record type | 46 |
| 5.1. | Differences between NSEC and NSEC3 | 47 |
| 5.2. | NSEC or NSEC3 | 47 |
| 5.3. | NSEC3 parameters | 48 |
| 5.3.1. | NSEC3 Algorithm | 48 |
| 5.3.2. | NSEC3 Iterations | 48 |
| 5.3.3. | NSEC3 Salt | 49 |
| 5.3.4. | Opt-out | 49 |
| 6. | Security Considerations | 49 |
| 7. | IANA considerations | 50 |
| 8. | Contributors and Acknowledgments | 50 |
| 9. | References | 51 |
| 9.1. | Normative References | 51 |

| | |
|--|----|
| 9.2. Informative References | 51 |
| Appendix A. Terminology | 53 |
| Appendix B. Typographic Conventions | 54 |
| Appendix C. Transition Figures for Special Case Algorithm Rollovers | 57 |
| Appendix D. Document Editing History | 61 |
| D.1. draft-ietf-dnsop-rfc4641-00 | 61 |
| D.2. version 0->1 | 62 |
| D.3. version 1->2 | 62 |
| D.4. version 2->3 | 63 |
| D.5. version 3->4 | 64 |
| D.6. version 4->5 | 64 |
| D.7. Subversion information | 64 |

1. Introduction

This document describes how to run a DNS Security (DNSSEC)-enabled environment. It is intended for operators who have knowledge of the DNS (see RFC 1034 [1] and RFC 1035 [2]) and want to deploy DNSSEC (RFC 4033 [3], RFC 4034 [4], and RFC 4035 [5]). The focus of the document is on serving authoritative DNS information and is aimed at zone owners, name server operators, registries, registrars and registrants. It assumes that there is no direct relation between those entities and the operators of validating recursive name servers (validators).

During workshops and early operational deployment, operators and system administrators have gained experience about operating the DNS with security extensions (DNSSEC). This document translates these experiences into a set of practices for zone administrators. At the time of writing -the root has just been signed and the first secure delegations are provisioned- there exists relatively little experience with DNSSEC in production environments below the TLD level; this document should therefore explicitly not be seen as representing 'Best Current Practices'. Instead, it describes the decisions that should be made when deploying DNSSEC, gives the choices available for each one, and provides some operational guidelines. The document does not give strong recommendations, that may be subject for a future version of this document. [OK: This is really a straw-man and causes a difference in tone that I believe was the instruction of the WG during the IETF 77 meeting. The document could be made much shorter when particular recommendations are made? Is there a general consensus that we should currently not make particular recommendations?]

The procedures herein are focused on the maintenance of signed zones (i.e., signing and publishing zones on authoritative servers). It is intended that maintenance of zones such as re-signing or key rollovers be transparent to any verifying clients.

The structure of this document is as follows. In Section 2, we discuss the importance of keeping the "chain of trust" intact. Aspects of key generation and storage of keys are discussed in Section 3; the focus in this section is mainly on the security of the private part of the key(s). Section 4 describes considerations concerning the public part of the keys. Since these public keys appear in the DNS one has to take into account all kinds of timing issues, which are discussed in Section 4.4. Section 4.1 and Section 4.2 deal with the rollover, or replacement, of keys. Finally, Section 4.3 discusses considerations on how parents deal with their children's public keys in order to maintain chains of trust.

The typographic conventions used in this document are explained in Appendix B.

Since this is a document with operational suggestions and there are no protocol specifications, the RFC 2119 [6] language does not apply.

This document [OK: when approved] obsoletes RFC 4641 [14].

[OK: Editorial comments and questions are indicated by square brackets and editor initials]

1.1. The Use of the Term 'key'

It is assumed that the reader is familiar with the concept of asymmetric keys on which DNSSEC is based (public key cryptography RFC4949 [15]). Therefore, this document will use the term 'key' rather loosely. Where it is written that 'a key is used to sign data' it is assumed that the reader understands that it is the private part of the key pair that is used for signing. It is also assumed that the reader understands that the public part of the key pair is published in the DNSKEY Resource Record and that it is the public part that is used in key exchanges.

1.2. Time Definitions

In this document, we will be using a number of time-related terms. The following definitions apply:

- o "Signature validity period" The period that a signature is valid. It starts at the time specified in the signature inception field of the RRSIG RR and ends at the time specified in the expiration field of the RRSIG RR.
- o "Signature publication period" Time after which a signature (made with a specific key) is replaced with a new signature (made with the same key) or removed. This replacement takes place by publishing the relevant RRSIG in the master zone file. After one stops publishing an RRSIG in a zone, it may take a while before the RRSIG has expired from caches and has actually been removed from the DNS.
- o "Key effectivity period" The period during which a key pair is expected to be effective. It is defined as the time between the first inception time stamp and the last expiration date of any signature made with this key, regardless of any discontinuity in the use of the key. The key effectivity period can span multiple signature validity periods.

- o "Maximum/Minimum Zone Time to Live (TTL)" The maximum or minimum value of the TTLs from the complete set of RRs in a zone. Note that the minimum TTL is not the same as the MINIMUM field in the SOA RR. See RFC2308 [9] for more information.

2. Keeping the Chain of Trust Intact

Maintaining a valid chain of trust is important because broken chains of trust will result in data being marked as Bogus (as defined in RFC4033 [3] Section 5), which may cause entire (sub)domains to become invisible to verifying clients. The administrators of secured zones need to realize that to verifying clients their zone is, part of a chain of trust.

As mentioned in the introduction, the procedures herein are intended to ensure that maintenance of zones, such as re-signing or key rollovers, will be transparent to the verifying clients on the Internet.

Administrators of secured zones will need to keep in mind that data published on an authoritative primary server will not be immediately seen by verifying clients; it may take some time for the data to be transferred to other (secondary) authoritative nameservers and clients may be fetching data from caching non-authoritative servers. In this light, note that the time for a zone transfer from master to slave can be negligible when using NOTIFY [8] and incremental transfer (IXFR) [7]. It increases when full zone transfers (AXFR) are used in combination with NOTIFY. It increases even more if you rely on full zone transfers based on only the SOA timing parameters for refresh.

For the verifying clients, it is important that data from secured zones can be used to build chains of trust regardless of whether the data came directly from an authoritative server, a caching nameserver, or some middle box. Only by carefully using the available timing parameters can a zone administrator ensure that the data necessary for verification can be obtained.

The responsibility for maintaining the chain of trust is shared by administrators of secured zones in the chain of trust. This is most obvious in the case of a 'key compromise' when a trade-off must be made between maintaining a valid chain of trust and replacing the compromised keys as soon as possible. Then zone administrators will have to decide, between keeping the chain of trust intact - thereby allowing for attacks with the compromised key - or deliberately breaking the chain of trust and making secured subdomains invisible to security-aware resolvers. (Also see Section 4.2.)

3. Keys Generation and Storage

This section describes a number of considerations with respect to the use of keys. For the design of a operational procedure for key generation and storage the a number of decisions need to be made:

- o Does one differentiate between Zone Signing and Key Signing Keys or is the use of one type of key sufficient?
- o Are Key Signing Keys (likely to be) in use as Trust Anchors?
- o What are the timing parameters that are allowed by the operational requirements?
- o What are the cryptographic parameters that fit the operational need?

The following section discusses the considerations that need to be taken into account when making those choices.

3.1. Operational Motivation for Zone Signing and Key Signing Keys

The DNSSEC validation protocol does not distinguish between different types of DNSKEYs. The motivations to differentiate between keys are purely operational; validators will not make a distinction.

For operational reasons, described below, it is possible to designate one or more keys as Key Signing Keys (KSKs). These keys will only sign the apex DNSKEY RRSset in a zone. Other keys can be used to sign all the RRSets in a zone that require signatures. They are referred to as Zone Signing Keys (ZSKs). In case the differentiation between KSK and ZSK is not made we talk about a Single Type signing scheme.

If the two functions are separated then, for almost any method of key management and zone signing, the KSK is used less frequently than the ZSK. Once a key set is signed with the KSK, all the keys in the key set can be used as ZSKs. If there has been an event that increases the risk that a ZSK is compromised it can be simply dropped from the key set. The new key set is then re-signed with the KSK.

Changing a key that is a a secure entry point (SEP) for a zone can be relatively expensive as it involves interaction with 3rd parties: When a key is only pointed to by a DS record in the parent zone, one needs to complete the interaction with the responsible registry and wait for the updated DS record to appear in the DNS. In the case where a key is configured as a trust-anchor one has to wait until one has sufficient confidence that all trust anchors have been replaced. In fact, it may be that one is not able to reach the complete user-

base with information about the key rollover.

There is also a risk that keys are compromised through theft or loss. For keys that are installed on file-systems of nameservers that are connected to the network (e.g. for dynamic updates) that risk is relatively high. Where keys are stored on Hardware Security Modules (HSMs) or stored off-line, such risk is relatively low. By separating the KSK and ZSK functionality these risks can be managed while making the tradeoff against the costs involved. For example, a KSK can be stored off-line or with more limitation on access control than ZSKs which need to be readily available for operational purposes such as the addition or deletion of zone data. For example, a KSK stored on a smartcard, that is kept in a safe, combined with a ZSK stored on a filesystem accessible by operators for daily routine may provide more operational flexibility and higher computational performance than a single key (with combined KSK and ZSK functionality) stored on an HSM.

Finally there is a risk of cryptanalysis of the key material. The costs of such analysis are correlated to the length of the key. However, cryptanalysis arguments provide no strong motivation for a KSK/ZSK split. Suppose one differentiates between a KSK and a ZSK whereby the KSK effectivity period is X times the ZSK effectivity period. Then, in order for the resistance to cryptanalysis to be the same for the KSK and the ZSK, the KSK needs to be X times stronger than the ZSK. Since for all practical purposes X will somewhere of the order of 10 to 100, the associated key sizes will vary only about a byte in size for symmetric keys. When translated to asymmetric keys, is still too insignificant a size difference to warrant a key-split; it only marginally affects the packet size and signing speed.

The arguments for differentiation between the ZSK and KSK are weakest when:

- o the exposure to risk is low (e.g. when keys are stored on HSMs);
- o one can be certain that a key is not used as a trust-anchor;
- o maintenance of the various keys cannot be performed through tools (is prone to human error); and
- o the interaction through the registrar-registry provisioning chain -- in particular the timely appearance of a new DS record in the parent zone in emergency situations -- is predictable.

If the above holds then the costs of the operational complexity of a KSK-ZSK split may outweigh the costs of operational flexibility and

choosing a single type signing scheme is a reasonable option. In other cases we advise that the separation between KSKs and ZSKs is made and that the SEP flag is exclusively set on KSKs.

3.2. Practical Consequences of KSK and ZSK Separation

Given the assumption that for KSKs the SEP flag is set, the KSK can be distinguished from a ZSK by examining the flag field in the DNSKEY RR: If the flag field is an odd number the RR is a KSK; otherwise it is a ZSK.

The Zone Signing Key can be used to sign all the data in a zone on a regular basis. When a Zone Signing Key is to be rolled, no interaction with the parent is needed. This allows for signature validity periods on the order of days.

The Key Signing Key is only to be used to sign the DNSKEY RRs in a zone. If a Key Signing Key is to be rolled, there will be interactions with parties other than the zone administrator. If there is a parent zone, these can include the registry of the parent zone or administrators of verifying resolvers that have the particular key configured as secure entry points. In the latter case, everyone relying on the trust anchor needs to roll over to the new key, a process that may be subject to stability costs if automated trust-anchor rollover mechanisms (such as e.g. RFC5011 [16]) are not in place. Hence, the key effectivity period of these keys can and should be made much longer.

3.2.1. Rolling a KSK that is not a trust-anchor

There are 3 schools of thought on rolling a KSK that is not a trust anchor:

- o It should be done frequently and regularly (possibly every few months) so that a key rollover remains an operational routine.
- o It should be done frequently but irregularly. Frequently meaning every few months, again based on the argument that a rollover is a practiced and common operational routine, and irregular meaning with a large jitter, so that 3rd parties do not start to rely on the key and will not be tempted to configure it as a trust-anchor.
- o It should only be done when it is known or strongly suspected that the key can be or has been compromised.

There is no widespread agreement on which of these three schools of thought is better for different deployments of DNSSEC. There is a stability cost every time a non-anchor KSK is rolled over, but it is

possibly low if the communication between the child and the parent is good. On the other hand, the only completely effective way to tell if the communication is good is to test it periodically. Thus, rolling a KSK with a parent is only done for two reasons: to test and verify the rolling system to prepare for an emergency, and in the case of (preventing) an actual emergency.

Finally, in most cases a zone owner cannot be fully certain that the zone's KSK is not in use as a trust-anchor somewhere. While the configuration of trust-anchors is not the responsibility of the zone owner there may be stability costs for the validator administrator that (wrongfully) configured the trust-anchor when the zone owner roles a KSK.

3.2.2. Rolling a KSK that is a trust-anchor

The same operational concerns apply to the rollover of KSKs that are used as trust-anchors: if a trust anchor replacement is done incorrectly, the entire domain that the trust anchor covers will become bogus until the trust anchor is corrected.

In a large number of cases it will be safe to work from the assumption that one's keys are not in use as trust-anchors. If a zone owner publishes a "DNSSEC Signing Policy and Practice Statement" [25] that should be explicit about the fact whether the existence of trust anchors will be taken into account in any way or not. There may be cases where local policies enforce the configuration of trust-anchors on zones which are mission critical (e.g. in enterprises where the trust-anchor for the enterprise domain is configured in the enterprise's validator) It is expected that the zone owners are aware of such circumstances.

One can argue that because of the difficulty of getting all users of a trust anchor to replace an old trust anchor with a new one, a KSK that is a trust anchor should never be rolled unless it is known or strongly suspected that the key has been compromised. In other words the costs of a KSK rollover are prohibitively high because some users cannot be reached.

However, the "operational habit" argument also applies to trust anchor reconfiguration at the clients' validators. If a short key effectivity period is used and the trust anchor configuration has to be revisited on a regular basis, the odds that the configuration tends to be forgotten is smaller. In fact, the costs for those users can be minimized by automating the rollover RFC5011 [16] and by rolling the key regularly (and advertising such) so that the operators of recursive nameservers will put the appropriate mechanism in place to deal with these stability costs, or, in other words,

budget for these costs instead of incurring them unexpectedly.

It is therefore recommended, to roll KSKs that are likely to be used as trust-anchors, on a regular basis if and only if those rollovers can be tracked using standardized (e.g. RFC5011) mechanisms.

3.2.3. The use of the SEP flag

The so-called Secure Entry Point (SEP) [5] flag can be used to distinguish between keys that are intended to be used as the secure entry point into the zone when building chains of trust, e.g they are (to be) pointed to by parental DS RRs or configured as a trust-anchor.

While the SEP flag does not play any role in the failure it is used in practice for operational purposes such as for the rollover mechanism described in RFC5011 [16]. The common convention is to set the SEP flag on any key that is used for key exchanges with the parent and/or potentially used for configuration as a trust anchor. Therefore it is recommended that the SEP flag is set on KSKs and not on ZSKs, while in those cases where a distinction between KSK and ZSK is not made (i.e. for a Single Type signing scheme) it is recommended that the SEP flag is set on all keys.

Note that signing tools may assume a KSK/ZSK split and use the (non) presence of the SEP flag to determine which key is to be used for signing zone data; these tools may get confused when a single type signing scheme is used.

3.3. Key Effectivity Period

In general the available key length sets an upper limit on the Key Effectivity Period. For all practical purposes it is sufficient to define the Key Effectivity Period based on purely operational requirements and match the key length to that value. Ignoring the operational perspective, a reasonable effectivity period for KSKs that have corresponding DS records in the parent zone is of the order of 2 decades or longer. That is, if one does not plan to test the rollover procedure, the key should be effective essentially forever, and only rolled over in case of emergency.

When one chooses for a regular key-rollover, a reasonable key effectivity period for KSKs that have a parent zone is 13 months, with the intent to replace them after 12 months. As argued above, this annual rollover gives operational practice of rollovers for both the zone and validator administrators. Besides, in most environments a year is a time-span that is easily planned and communicated.

Where keys are stored on on-line systems and the exposure to various threats of compromise is fairly high, an intended key effectivity period of a month is reasonable for Zone Signing Keys.

Although key effectivity periods can be made very short -as in a few minutes- when replacing keys one has to take into account the considerations from Section 4.4 and Section 4.1.

The motivation for having the ZSK's effectivity period shorter than the KSK's effectivity period is rooted in the operational consideration that it is more likely that operators have more frequent read access to the ZSK than to the KSK. If ZSK's are maintained on cryptographic Hardware Security Modules (HSM) than the motivation to have different key effectivity periods is weakend.

In fact, if the risk of loss, theft or other compromise is the same for a zone and key signing key there is little reason to choose different effectivity periods for ZSKs and KSKs. And when the split between ZSKs and KSKs is not made, the argument is redundant.

There are certainly cases (e.g. where the the costs and risk of compromise, and the costs and risks involved with having to perform an emergency roll are also low) that the use of a single type signing scheme with a long key effectivity period is a good choice.

3.4. Cryptographic Considerations

3.4.1. Key Algorithm

There are currently two types of signature algorithms that can be used in DNSSEC: RSA and DSA. Both are fully specified in many freely-available documents, and both are widely considered to be patent-free. The creation of signatures with RSA and DSA takes roughly the same time, but DSA is about ten times slower for signature verification.

We suggest the use of RSA/SHA-256 as the preferred signature algorithms and RSA/SHA-1 as an alternative. Both have advantages and disadvantages. RSA/SHA-1 has been deployed for many years, while RSA/SHA-256 has only begun to be deployed. On the other hand, it is expected that if effective attacks on either algorithm appear, they will appear for RSA/SHA-1 first. RSA/MD5 should not be considered for use because RSA/MD5 will very likely be the first common-use signature algorithm to have an effective attack.

At the time of publication, it is known that the SHA-1 hash has cryptanalysis issues and work is in progress on addressing them. We recommend the use of public key algorithms based on hashes stronger

than SHA-1 (e.g., SHA-256) as soon as these algorithms are available in implementations (see RFC5702 [23] and RFC4509 [20]).

3.4.2. Key Sizes

DNSSEC signing keys should be large enough to avoid all known cryptographic attacks during the effectivity period of the key. To date, despite huge efforts, no one has broken a regular 1024-bit key; in fact, the best completed attack is estimated to be the equivalent of a 700-bit key. An attacker breaking a 1024-bit signing key would need to expend phenomenal amounts of networked computing power in a way that would not be detected in order to break a single key. Because of this, it is estimated that most zones can safely use 1024-bit keys for at least the next ten years. (A 1024-bit asymmetric key has an approximate equivalent strength of a symmetric 80-bit key.)

Owners of keys that are used as extremely high value trust anchors, or non-anchor keys that may be difficult to roll over, may want to use lengths longer than 1024 bits. Typically, the next larger key size used is 2048 bits, which has the approximate equivalent strength of a symmetric 112-bit key (e.g. RFC3766 [12]). In a standard CPU, it takes about four times as long to sign or verify with a 2048-bit key as it does with a 1024-bit key.

Another way to decide on the size of key to use is to remember that the effort it takes for an attacker to break a 1024-bit key is the same regardless of how the key is used. If an attacker has the capability of breaking a 1024-bit DNSSEC key, he also has the capability of breaking one of the many 1024-bit TLS trust anchor keys that are currently installed in web browsers. If the value of a DNSSEC key is lower to the attacker than the value of a TLS trust anchor, the attacker will use the resources to attack the latter.

It is possible that there will be an unexpected improvement in the ability for attackers to break keys, and that such an attack would make it feasible to break 1024-bit keys but not 2048-bit keys. If such an improvement happens, it is likely that there will be a huge amount of publicity, particularly because of the large number of 1024-bit TLS trust anchors build into popular web browsers. At that time, all 1024-bit keys (both ones with parent zones and ones that are trust anchors) can be rolled over and replaced with larger keys.

Earlier documents (including the previous version of this document) urged the use of longer keys in situations where a particular key was "heavily used". That advice may have been true 15 years ago, but it is not true today when using RSA or DSA algorithms and keys of 1024 bits or higher.

3.4.3. Private Key Storage

It is recommended that, where possible, zone private keys and the zone file master copy that is to be signed be kept and used in off-line, non-network-connected, physically secure machines only. Periodically, an application can be run to add authentication to a zone by adding RRSIG and NSEC/NSEC3 RRs. Then the augmented file can be transferred.

When relying on dynamic update [10] to manage a signed zone, be aware that at least one private key of the zone will have to reside on the master server (or reside on an HSM to which the server has access). This key is only as secure as the amount of exposure the server receives to unknown clients and the security of the host. Although not mandatory, one could administer a zone using a "hidden master" scheme that minimize the risk. In this arrangement the master that processes the dynamic updates is unavailable from general hosts on the Internet; it is not listed in the NS RRSets, although its name appears in the SOA RRs MNAME field. The nameservers in the NS RRSets are able to receive zone updates through IXFR, AXFR, or an out-of-band distribution mechanism, possibly in combination with NOTIFY or another mechanism to trigger zone replication.

The ideal situation is to have a one-way information flow to the network to avoid the possibility of tampering from the network. Keeping the zone master on-line on the network and simply cycling it through an off-line signer does not do this. The on-line version could still be tampered with if the host it resides on is compromised. For maximum security, the master copy of the zone file should be off-net and should not be updated based on an unsecured network mediated communication.

The ideal situation may not be achievable because of economic tradeoffs between risks and costs. For instance, keeping a zone file off-line is not practical and will increase the costs of operating a DNS zone. So in practice the machines on which zone files are maintained will be connected to a network. Operators are advised to take security measures to shield unauthorized access to the master copy in order to prevent modification of DNS data before its signed.

Similarly the choice for storing a private key in a HSM will be influenced by a tradeoff between various concerns:

- o The risks that an unauthorized person has unnoticed read-access to the private key
- o The remaining window of opportunity for the attacker.

- o The economic impact of the possible attacks (for a TLD that impact will typically be higher than for an individual users).
- o The costs of rolling the (compromised) keys. (The costs of rolling a ZSK is lowest and the costs of rolling a KSK that is in wide use as a trust anchor is highest.)
- o The costs of buying and maintaining an HSM.

For dynamically updated secured zones [10], both the master copy and the private key that is used to update signatures on updated RRs will need to be on-line.

3.4.4. Key Generation

Careful generation of all keys is a sometimes overlooked but is an absolutely essential element in any cryptographically secure system. The strongest algorithms used with the longest keys are still of no use if an adversary can guess enough to lower the size of the likely key space so that it can be exhaustively searched. Technical suggestions for the generation of random keys will be found in RFC 4086 [13] and NIST SP 800-900 [19]. In particular, one should carefully assess whether the random number generator used during key generation adheres to these suggestions.

Keys with a long effectivity period are particularly sensitive as they will represent a more valuable target and be subject to attack for a longer time than short-period keys. It is strongly recommended that long-term key generation occur off-line in a manner isolated from the network via an air gap or, at a minimum, high-level secure hardware.

3.4.5. Differentiation for 'High-Level' Zones?

In an earlier version of this document (RFC4641 [14]) we made a differentiation between key lengths for KSKs used for zones that are high in the DNS hierarchy and those for KSKs used low down.

This distinction is now considered not relevant. Longer key lengths for keys higher in the hierarchy are not useful because the cryptographic guidance is that everyone should use keys that no one can break. Also, it is impossible to judge which zones are more or less valuable to an attacker. An attack can only take place if the key compromise goes unnoticed and the attacker can act as a man-in-the-middle (MITM). For example if example.com is compromised and the attacker forges answers for somebank.example.com. and sends them out during an MITM, when the attack is discovered it will be simple to prove that example.com has been compromised and the KSK will be

rolled. Designing a long-term successful attack is difficult for keys at any level.

4. Signature Generation, Key Rollover, and Related Policies

4.1. Key Rollovers

Regardless of whether a zone uses periodic key rollovers in order to practice for emergencies, or only rolls over keys in an emergency, key rollovers are a fact of life when using DNSSEC. Zone administrators who are in the process of rolling their keys have to take into account that data published in previous versions of their zone still lives in caches. When deploying DNSSEC, this becomes an important consideration; ignoring data that may be in caches may lead to loss of service for clients.

The most pressing example of this occurs when zone material signed with an old key is being validated by a resolver that does not have the old zone key cached. If the old key is no longer present in the current zone, this validation fails, marking the data "Bogus". Alternatively, an attempt could be made to validate data that is signed with a new key against an old key that lives in a local cache, also resulting in data being marked "Bogus".

4.1.1. Zone Signing Key Rollovers

If the choice for splitting zone and key signing keys has been made then those two types of keys can be rolled separately and zone signing keys can be rolled without taking into account DS records from the parent or the configuration of such a key as trust-anchor.

For "Zone Signing Key rollovers", there are two ways to make sure that during the rollover data still cached can be verified with the new key sets or newly generated signatures can be verified with the keys still in caches. One schema, described in Section 4.1.1.2, uses double signatures; the other uses key pre-publication (Section 4.1.1.1). The pros, cons, and recommendations are described in Section 4.1.1.3.

4.1.1.1. Pre-Publish Zone Signing Key Rollover

This section shows how to perform a ZSK rollover without the need to sign all the data in a zone twice -- the "pre-publish key rollover". This method has advantages in the case of a key compromise. If the old key is compromised, the new key has already been distributed in the DNS. The zone administrator is then able to quickly switch to the new key and remove the compromised key from the zone. Another major advantage is that the zone size does not double, as is the case

with the double signature ZSK rollover.

Pre-publish key rollover involves four stages as follows:

| initial | new DNSKEY | new RRSIGs |
|---|--|--|
| SOA0 RRSIG_Z_10(SOA) | SOA1 RRSIG_Z_10(SOA) | SOA2 RRSIG_Z_11(SOA) |
| DNSKEY_K_1 DNSKEY_Z_10 | DNSKEY_K_1 DNSKEY_Z_10 DNSKEY_Z_11 | DNSKEY_K_1 DNSKEY_Z_10 DNSKEY_Z_11 |
| RRSIG_K_1(DNSKEY) RRSIG_Z_10(DNSKEY) | RRSIG_K_1(DNSKEY) RRSIG_Z_10(DNSKEY) | RRSIG_K_1(DNSKEY) RRSIG_Z_11(DNSKEY) |
| ----- | | |
| DNSKEY removal | | |
| ----- | | |
| SOA3 RRSIG_Z_11(SOA) | | |
| DNSKEY_K_1 DNSKEY_Z_11 | | |
| RRSIG_K_1(DNSKEY) RRSIG_Z_11(DNSKEY) | | |
| ----- | | |

Figure 1: Pre-Publish Key Rollover

initial: Initial version of the zone: DNSKEY_K_1 is the Key Signing Key. DNSKEY_Z_10 is used to sign all the data of the zone, the Zone Signing Key.

new DNSKEY: DNSKEY_Z_11 is introduced into the key set. Note that no signatures are generated with this key yet, but this does not secure against brute force attacks on the public key. The minimum duration of this pre-roll phase is the time it takes for the data to propagate to the authoritative servers plus TTL value of the key set.

new RRSIGs: At the "new RRSIGs" stage (SOA serial 2), DNSKEY_Z_11 is used to sign the data in the zone exclusively (i.e., all the signatures from DNSKEY_Z_10 are removed from the zone). DNSKEY_Z_10 remains published in the key set. This way data that

was loaded into caches from version 1 of the zone can still be verified with key sets fetched from version 2 of the zone. The minimum time that the key set including DNSKEY_Z_10 is to be published is the time that it takes for zone data from the previous version of the zone to expire from old caches, i.e., the time it takes for this zone to propagate to all authoritative servers plus the Maximum Zone TTL value of any of the data in the previous version of the zone.

DNSKEY removal: DNSKEY_Z_10 is removed from the zone. The key set, now only containing DNSKEY_K_1 and DNSKEY_Z_11, is re-signed with the DNSKEY_K_1 and DNSKEY_Z_11.

The above scheme can be simplified by always publishing the "future" key immediately after the rollover. The scheme would look as follows (we show two rollovers); the future key is introduced in "new DNSKEY" as DNSKEY_Z_12 and again a newer one, numbered 13, in "new DNSKEY (II)":

| initial | new RRSIGs | new DNSKEY |
|--------------------|--------------------|--------------------|
| SOA0 | SOA1 | SOA2 |
| RRSIG_Z_10(SOA) | RRSIG_Z_11(SOA) | RRSIG_Z_11(SOA) |
| DNSKEY_K_1 | DNSKEY_K_1 | DNSKEY_K_1 |
| DNSKEY_Z_10 | DNSKEY_Z_10 | DNSKEY_Z_11 |
| DNSKEY_Z_11 | DNSKEY_Z_11 | DNSKEY_Z_12 |
| RRSIG_K_1(DNSKEY) | RRSIG_K_1 (DNSKEY) | RRSIG_K_1(DNSKEY) |
| RRSIG_Z_10(DNSKEY) | RRSIG_Z_11(DNSKEY) | RRSIG_Z_11(DNSKEY) |
| ----- | | |
| ----- | | |
| new RRSIGs (II) | new DNSKEY (II) | |
| SOA3 | SOA4 | |
| RRSIG_Z_12(SOA) | RRSIG_Z_12(SOA) | |
| DNSKEY_K_1 | DNSKEY_K_1 | |
| DNSKEY_Z_11 | DNSKEY_Z_12 | |
| DNSKEY_Z_12 | DNSKEY_Z_13 | |
| RRSIG_K_1(DNSKEY) | RRSIG_K_1(DNSKEY) | |
| RRSIG_Z_12(DNSKEY) | RRSIG_Z_12(DNSKEY) | |
| ----- | | |

Figure 2: Pre-Publish Zone Signing Key Rollover, Showing Two Rollovers

Note that the key introduced in the "new DNSKEY" phase is not used for production yet; the private key can thus be stored in a physically secure manner and does not need to be 'fetched' every time a zone needs to be signed.

4.1.1.2. Double Signature Zone Signing Key Rollover

This section shows how to perform a ZSK key rollover using the double zone data signature scheme, aptly named "double signature rollover".

During the "new DNSKEY" stage the new version of the zone file will need to propagate to all authoritative servers and the data that exists in (distant) caches will need to expire, requiring at least the Maximum Zone TTL.

Double signature ZSK rollover involves three stages as follows:

| initial | new DNSKEY | DNSKEY removal |
|--------------------|--------------------|--------------------|
| SOA0 | SOA1 | SOA2 |
| RRSIG_Z_10(SOA) | RRSIG_Z_10(SOA) | RRSIG_Z_11(SOA) |
| | RRSIG_Z_11(SOA) | |
| DNSKEY_K_1 | DNSKEY_K_1 | DNSKEY_K_1 |
| DNSKEY_Z_10 | DNSKEY_Z_10 | DNSKEY_Z_11 |
| | DNSKEY_Z_11 | |
| RRSIG_K_1(DNSKEY) | RRSIG_K_1(DNSKEY) | RRSIG_K_1(DNSKEY) |
| RRSIG_Z_10(DNSKEY) | RRSIG_Z_10(DNSKEY) | RRSIG_Z_11(DNSKEY) |
| | RRSIG_Z_11(DNSKEY) | |

Figure 3: Double Signature Zone Signing Key Rollover

initial: Initial Version of the zone: DNSKEY_K_1 is the Key Signing Key. DNSKEY_Z_10 is used to sign all the data of the zone, the Zone Signing Key.

new DNSKEY: At the "New DNSKEY" stage (SOA serial 1) DNSKEY_Z_11 is introduced into the key set and all the data in the zone is signed with DNSKEY_Z_10 and DNSKEY_Z_11. The rollover period will need to continue until all data from version 0 of the zone has expired from remote caches. This will take at least the Maximum Zone TTL of version 0 of the zone.

DNSKEY removal: DNSKEY_Z_10 is removed from the zone. All the signatures from DNSKEY_Z_10 are removed from the zone. The key set, now only containing DNSKEY_Z_11, is re-signed with DNSKEY_K_1 and DNSKEY_Z_11.

At every instance, RRSIGs from the previous version of the zone can be verified with the DNSKEY RRSet from the current version and the other way around. The data from the current version can be verified with the data from the previous version of the zone. The duration of the "new DNSKEY" phase and the period between rollovers should be at least the Maximum Zone TTL.

Making sure that the "new DNSKEY" phase lasts until the signature expiration time of the data in the initial version of the zone is recommended. This way all caches are cleared of the old signatures. However, this duration could be considerably longer than the Maximum Zone TTL, making the rollover a lengthy procedure.

Note that in this example we assumed that the zone was not modified during the rollover. New data can be introduced in the zone as long as it is signed with both keys.

4.1.1.3. Pros and Cons of the Schemes

Pre-publish key rollover: This rollover does not involve signing the zone data twice. Instead, before the actual rollover, the new key is published in the key set and thus is available for cryptanalysis attacks. A small disadvantage is that this process requires four steps. Also the pre-publish scheme involves more parental work when used for KSK rollovers as explained in Section 4.1.3.

Double signature ZSK rollover: The drawback of this signing scheme is that during the rollover the number of signatures in your zone doubles; this may be prohibitive if you have very big zones. An advantage is that it only requires three steps.

4.1.2. Key Signing Key Rollovers

For the rollover of a Key Signing Key, the same considerations as for the rollover of a Zone Signing Key apply. However, we can use a double signature scheme to guarantee that old data (only the apex key set) in caches can be verified with a new key set and vice versa. Since only the key set is signed with a KSK, zone size considerations do not apply.

| initial | new DNSKEY | DS change | DNSKEY removal |
|--------------------|--------------------|----------------|--------------------|
| Parent: | | | |
| SOA0 | -----> | SOA1 | -----> |
| RRSIG_par(SOA) | -----> | RRSIG_par(SOA) | -----> |
| DS_K_1 | -----> | DS_K_2 | -----> |
| RRSIG_par(DS) | -----> | RRSIG_par(DS) | -----> |
| Child: | | | |
| SOA0 | SOA1 | -----> | SOA2 |
| RRSIG_Z_10(SOA) | RRSIG_Z_10(SOA) | -----> | RRSIG_Z_10(SOA) |
| | | -----> | |
| DNSKEY_K_1 | DNSKEY_K_1 | -----> | DNSKEY_K_2 |
| | DNSKEY_K_2 | -----> | |
| DNSKEY_Z_10 | DNSKEY_Z_10 | -----> | DNSKEY_Z_10 |
| RRSIG_K_1(DNSKEY) | RRSIG_K_1 (DNSKEY) | -----> | RRSIG_K_2(DNSKEY) |
| | RRSIG_K_2 (DNSKEY) | -----> | |
| RRSIG_Z_10(DNSKEY) | RRSIG_Z_10(DNSKEY) | -----> | RRSIG_Z_10(DNSKEY) |

Figure 4: Stages of Deployment for a Double Signature Key Signing Key Rollover

initial: Initial version of the zone. The parental DS points to DNSKEY_K_1. Before the rollover starts, the child will have to verify what the TTL is of the DS RR that points to DNSKEY_K_1 -- it is needed during the rollover and we refer to the value as TTL_DS.

new DNSKEY: During the "new DNSKEY" phase, the zone administrator generates a second KSK, DNSKEY_K_2. The key is provided to the parent, and the child will have to wait until a new DS RR has been generated that points to DNSKEY_K_2. After that DS RR has been published on all servers authoritative for the parent's zone, the zone administrator has to wait at least TTL_DS to make sure that the old DS RR has expired from caches.

DS change: The parent replaces DS_K_1 with DS_K_2.

DNSKEY removal: DNSKEY_K_1 has been removed.

The scenario above puts the responsibility for maintaining a valid chain of trust with the child. It also is based on the premise that the parent only has one DS RR (per algorithm) per zone. An alternative mechanism has been considered. Using an established

trust relation, the interaction can be performed in-band, and the removal of the keys by the child can possibly be signaled by the parent. In this mechanism, there are periods where there are two DS RRs at the parent. Since at the moment of writing the protocol for this interaction has not been developed, further discussion is out of scope for this document.

4.1.2.1. Special Considerations for RFC5011 KSK rollover

The scenario sketched above assumes that the KSK is not in use as a trust-anchor too but that validating nameservers exclusively depend on the parental DS record to establish the zone's security. If it is known that validating nameservers have configured trust-anchors then such needs to be taken into account. Here we assume that operators of zones will deploy RFC5011 [16] style rollovers.

RFC5011 style rollovers increase the duration of key rollovers: the key to be removed must first be revoked. Thus, before the DNSKEY_K_1 removal phase, DNSKEY_K_1 must be published for one more Maximum Zone TTL with the REVOKE bit set. The revoked key must be self-signed, so in this phase the DNSKEY RRset must also be signed with DNSKEY_K_1.

4.1.3. Difference Between ZSK and KSK Rollovers

Note that KSK rollovers and ZSK rollovers are different in the sense that a KSK rollover requires interaction with the parent (and possibly replacing of trust anchors) and the ensuing delay while waiting for it.

A zone key rollover can be handled in two different ways: pre-publish (Section 4.1.1.1) and double signature (Section 4.1.1.2).

As the KSK is used to validate the key set and because the KSK is not changed during a ZSK rollover, a cache is able to validate the new key set of the zone. The pre-publish method would also work for a KSK rollover. The records that are to be pre-published are the parental DS RRs. The pre-publish method has some drawbacks for KSKs. We first describe the rollover scheme and then indicate these drawbacks.

| initial | new DS | new DNSKEY | DS/DNSKEY removal |
|--------------------|----------------|--------------------|-------------------|
| Parent: | | | |
| SOA0 | SOA1 | -----> | SOA2 |
| RRSIG_par(SOA) | RRSIG_par(SOA) | -----> | RRSIG_par(SOA) |
| DS_K_1 | DS_K_1 | -----> | DS_K_2 |
| | DS_K_2 | -----> | |
| RRSIGpar(DS) | RRSIG_par(DS) | -----> | RRSIG_par(DS) |
| Child: | | | |
| SOA0 | -----> | SOA1 | SOA1 |
| RRSIG_Z_10(SOA) | -----> | RRSIG_Z_10(SOA) | RRSIG_Z_10(SOA) |
| | -----> | | |
| DNSKEY_K_1 | -----> | DNSKEY_K_2 | DNSKEY_K_2 |
| | -----> | | |
| DNSKEY_Z_10 | -----> | DNSKEY_Z_10 | DNSKEY_Z_10 |
| RRSIG_K_1 (DNSKEY) | -----> | RRSIG_K_2(DNSKEY) | RRSIG2 (DNSKEY) |
| RRSIG_Z_10(DNSKEY) | -----> | RRSIG_Z_10(DNSKEY) | RRSIG10(DNSKEY) |

Figure 5: Stages of Deployment for a Pre-Publish Key Signing Key Rollover

When the child zone wants to roll, it notifies the parent during the "new DS" phase and submits the new key (or the corresponding DS) to the parent. The parent publishes DS_K_1 and DS_K_2, pointing to DNSKEY_K_1 and DNSKEY_K_2, respectively. During the rollover ("new DNSKEY" phase), which can take place as soon as the new DS set propagated through the DNS, the child replaces DNSKEY_K_1 with DNSKEY2. Immediately after that ("DS/DNSKEY removal" phase), it can notify the parent that the old DS record can be deleted.

The drawbacks of this scheme are that during the "new DS" phase the parent cannot verify the match between the DS_K_2 RR and DNSKEY_K_2 using the DNS -- as DNSKEY_K_2 is not yet published. Besides, we introduce a "security lame" key (see Section 4.3.3). Finally, the child-parent interaction consists of two steps. The "double signature" method only needs one interaction.

4.1.4. Rollover for a Single Type Signing Key rollover

The rollover of a DNSKEY when a Single Type Signing scheme is used is subject to the same requirement as the rollover of a KSK or ZSK: During any stage of the rollover the chain of trust needs to continue to validate for any combination of data in the zone as well as data that may still live in distant caches.

There are two variants for this rollover. Since the choice for a Single Type Signing scheme is motivated by operational simplicity we first describe the most straightforward rollover scheme first.

| initial | new DNSKEY | DS change | DNSKEY removal |
|-------------------|-------------------|----------------|-------------------|
| Parent: | | | |
| SOA0 | -----> | SOA1 | -----> |
| RRSIG_par(SOA) | -----> | RRSIG_par(SOA) | -----> |
| DS1 | -----> | DS2 | -----> |
| RRSIG_par(DS) | -----> | RRSIG_par(DS) | -----> |
| Child: | | | |
| SOA0 | SOA1 | -----> | SOA2 |
| RRSIG_S_1(SOA) | RRSIG_S_1(SOA) | -----> | RRSIG_S_2(SOA) |
| | RRSIG_S_2(SOA1) | -----> | |
| DNSKEY_S_1 | DNSKEY_S_1 | -----> | DNSKEY_S_2 |
| | DNSKEY_S_2 | -----> | |
| RRSIG_S_1(DNSKEY) | RRSIG_S_1(DNSKEY) | -----> | RRSIG_S_2(DNSKEY) |
| | RRSIG_S_2(DNSKEY) | -----> | |

Figure 6: Stages of the Straightforward rollover in a Single Type Signing scheme

initial: Parental DS points to DNSKEY_K_1. All RR sets in the zone are signed with DNSKEY_K_1.

new DNSKEY: A new key (DNSKEY_K_2) is introduced and all the RR sets are signed with both DNSKEY_K_1 and DNSKEY_K_2.

DS change: After the DNSKEY RRset with the two keys had time to propagate into distant caches (that is the key set exclusively containing DNSKEY_K_1 has been expired) the parental DS record can be changed.

DNSKEY removal: After the DS RRset containing DS_K_1 has expired from distant caches DNSKEY_K_1 can be removed from the DNSKEY RRset .

There is a second variety of this rollover during which one introduces a new DNSKEY into the key set and signs the keyset with both keys while signing the zone data with only the original DNSKEY_K_1. One replaces the DNSKEY_K_1 signatures with signatures made with DNSKEY_K_2 at the moment of DNSKEY_K_1 removal.

The second variety of this rollover can be considered when zone size

considerations prevent the introduction of double signatures over all of the zone data although in that case choosing for a KSK/ZSK split may be a better option.

A double DS rollover scheme is compatible with a rollover using a Single Type signing scheme although in order to maintain a valid chain of trust the zone data would need to be published with a double signatures or a double key key set would need to be published. Since this leads to increase in zone and packet size at both child and parent there are little benefits to a double DS rollover with a Single Type signing scheme.

4.1.5. Algorithm rollovers

A special class of key rollover is the one needed for a change of key algorithms (either adding a new algorithm, removing an old algorithm, or both). Additional steps are needed to retain integrity during this rollover. We first describe the generic case, special considerations for rollovers that involve trust-anchors, single type keys, and rollovers from NSEC to NSEC3 are discussed below.

Because of the algorithm downgrade protection in RFC4035 section 2.2, you may not have a key of an algorithm for which you do not have signatures, and you may not have a DS record in the parent zone of an algorithm for which you don't have a corresponding key in the zone apex.

When adding a new algorithm, the signatures should be added first. After the TTL of RRSIGS has expired, and caches have dropped the old data covered by those signatures, the DNSKEY with the new algorithm can be added.

After the new algorithm has been added, the DS record can be exchanged using Double Signature Key Rollover. You cannot use Pre-publish key rollover method when you do key algorithm rollover.

When removing an old algorithm, the DNSKEY should be removed first, but only after the DS for the old algorithm was removed from the parent zone.

The following figure describes the steps. Whereby the trailing underscored number indicates the algorithm and ZSK and KSK indicate the obvious difference in key use. For example DNSKEY_KSK_1 is a the DNSKEY RR representing the public part of the old key signing key of algorithm type 1 while RRSIG_ZSK_2(SOA3) is the RRSIG RR made with the private part of the new zone signing key of algorithm type 2 over a SOA RR (that has serial number 3). It is assumed that the key that signs the SOA RR also signs all other non-DNSKEY RRset data.

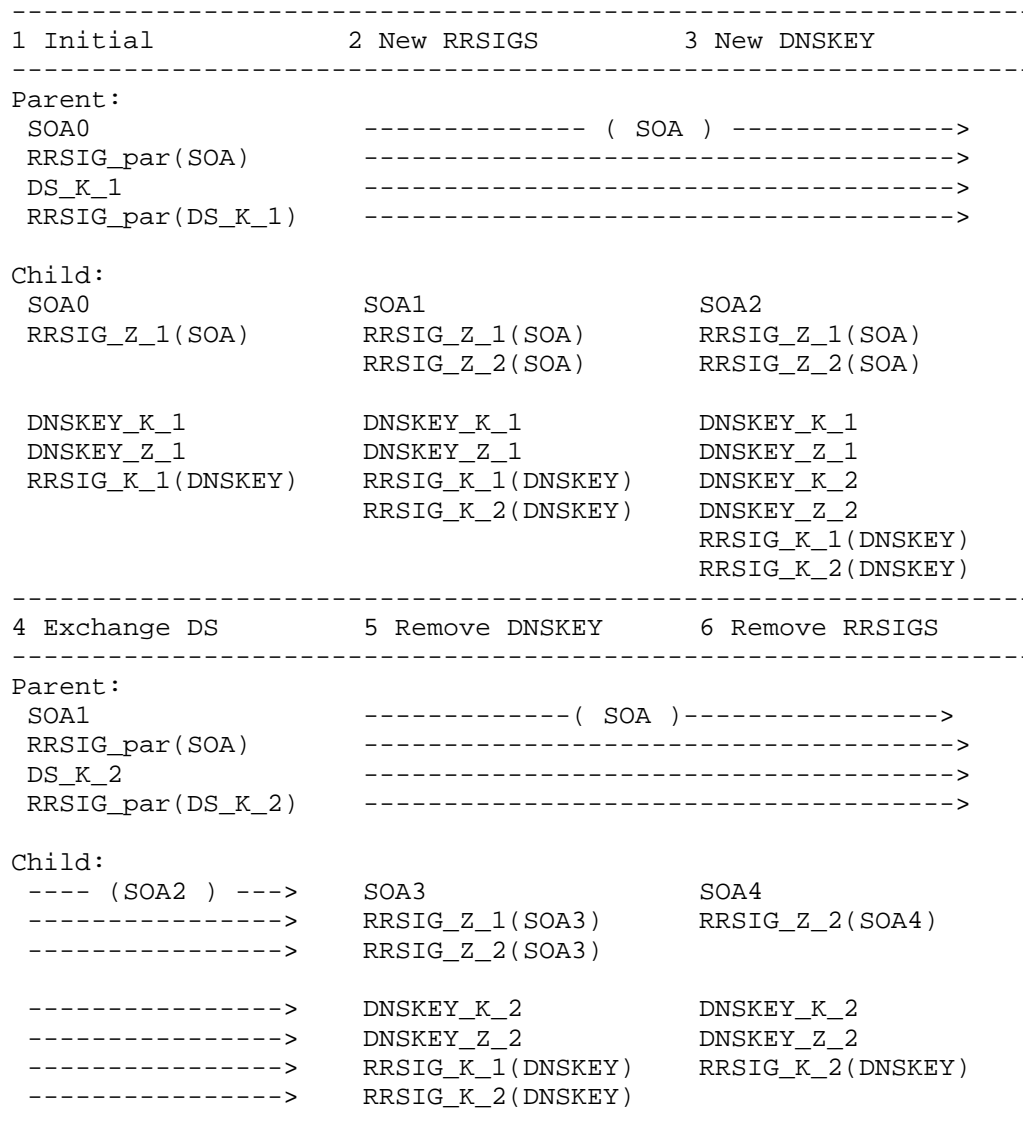


Figure 7: Stages of Deployment during an Algorithm Rollover

Step 1 describes state of the zone before any transition is done. Number of the keys may vary, but the algorithm of keys in the zone is same for all DNSKEY records.

Step 2: the signatures made with the new key over all records in the

zone are added, but the key itself is not. This includes the signature for the DNSKEY rrset. While in theory, the signatures of the keyset should always be synchronized with the keyset itself, it can be possible that RRSIGs are requested separately, so it is prudent to also sign the DNSKEY set with the new signature.

This step is needed to propagate the signatures created with the new algorithm to the caches. If you do not do that, it might happen that the resolver picks up the new DNSKEY RRset (with the new algorithm included), but still have the old list of signatures stored.

Step 3: After the cache data has expired, the new key can be added to the zone.

Step 4: After the cache data for the DNSKEY has expired, the DS record for the new key can be added to the parent zone and the DS record for the old key can be removed in the same step.

Step 5: After the cache data for the DS has expired, the old algorithm can be removed. This time the key needs to be removed first, before removing the signatures. The key is removed in this step, and after the cache data for the DNSKEY has expired, the signatures can also be removed during this step.

Below we deal with a few special cases of algorithm rollovers.

- 1: NSEC to NSEC3 : the case where one wants to roll algorithm in order to deploy NSEC3 instead of NSEC (Section 4.1.5.1).
- 2: Single Type Signing Scheme Algorithm Rollover : when you have chosen not to differentiate between Zone and Key signing keys (Section 4.1.5.2)
- 3: RFC5011 : when trust-anchors can track the roll via RFC5011 style rollover (Section 4.1.5.3)
- 4: 2 and 3 combined : when a Single Type Signing Scheme is rolled via RFC5011 (Section 4.1.5.4)

In addition to the narrative below these special cases are represented in the figures in Appendix C.

4.1.5.1. NSEC to NSEC3 algorithm rollover

A special case is the rollover from an NSEC signed zone to an NSEC3 signed zone. In this case algorithm numbers are used to signal support for NSEC3 but they do not mandate the use of NSEC3. Therefore NSEC records should remain in the zone until the rollover

to a new algorithm has completed and the new DNSKEY RR set has populated distant caches (at least one TTL into stage 4, or at any time during stage 5). At that point the validators that have not implemented NSEC3 will treat the zone as unsecured as soon as they follow the chain of trust to DS that points to a DNSKEY of the new algorithm while validators that support NSEC3 will happily validate using NSEC. Turning on NSEC3 can then be done when changing from zone serial number, realizing that that involves a resigning of the zone and the introduction of the NSECPARAM record in order to signal authoritative servers to start serving NSEC3 authenticated denial of existence.

Summarizing, an NSEC to NSEC3 rollover is an ordinary algorithm rollover whereby NSEC is used all the time and only after that rollover finished NSEC3 needs to be deployed.

4.1.5.2. Single Type Signing Scheme Algorithm Rollover

If one key is used that acts both as ZSK and KSK, the same scheme and figure as above applies whereby all DNSKEY_Z_* records from the table are removed and all RRSIG_Z_* are replaced with RRSIG_K_*. The requirement to sign with both algorithms and make sure that old RRSIGs had the opportunity to expire from distant caches before introducing the new algorithm in the DNSKEY RRset still holds.

Also see Figure 11 in Appendix C.

4.1.5.3. Algorithm rollover, RFC5011 style

Trust anchor algorithm rollover is as simple as a regular RFC5011 based rollover. However, the old trust anchor must be revoked before it is removed from the zone.

Take a look at the Figure 7 above. After Step 4 (Exchange DS) we need an additional step 4a whereby the DNSKEY is revoked (Revoke DNSKEY):

```

-----
4b Revoke DNSKEY
-----
Parent:
------(SOA)----->
----->
----->
----->

Child:
SOA3
RRSIG_Z_1(SOA3)
RRSIG_Z_2(SOA3)

DNSKEY_K_1_REVOKED
DNSKEY_Z_1
DNSKEY_K_2
DNSKEY_Z_2
RRSIG_K_1_REVOKED(DNSKEY)
RRSIG_K_2(DNSKEY)
-----

```

Figure 8: The Revoke DNSKEY state that is added to an algorithm rollover when RFC5011 is in use.

Also see Figure 12 in Appendix C.

4.1.5.4. Single Signing Type, RFC5011 style rollovers

Combining Single Signing Type and RFC5011 style rollovers is not trivial.

Should you choose to perform an RFC5011 style rollover with a Single Signing Type key then remember that section .1, RFC 5011 states:

Once the resolver sees the REVOKE bit, it MUST NOT use this key as a trust anchor or for any other purpose except to validate the RRSIG it signed over the DNSKEY RRSet specifically for the purpose of validating the revocation.

This means that if you revoke DNSKEY_KSK_1, it cannot be used to validate its signatures over non-DNSKEY RRsets. Thus, those RRsets should be signed with a shadow key, DNSKEY_ZSK_1, during the algorithm rollover. This shadow key can be introduced at the same time the signatures are pre-published, in step 2 (new RRSIGs). The shadow key must be removed at the same time the revoked KSK_1 is

removed from the zone. De-facto you temporarily falling back to a zone and key signing key model.

In other words, the rule that at every RRset there must be at least one signature for each algorithm used in the DNSKEY RRset still applies. This means that a different key with the same algorithm, other than the revoked key, must sign the entire zone. This can be the ZSK. More operations is needed if the single type signing scheme is used. Before rolling the algorithm, a new key must be introduced with the same algorithm as the key that is candidate for revocation. That key can then temporarily act as ZSK during the algorithm rollover.

There is one exception to the rule above. While all zone data must be signed with an unrevoked key, it is permissible to sign the keyset with a revoked key. The somewhat esoteric argument follows.

Resolvers that do not understand the RFC5011 Revoke flag will handle DNSKEY_K_1_REVOKED the same as a new key DNSKEY_K_*. The identity of the public key is partly determined by the flag field and by setting REVOKE the identity changed. Resolvers that implement RFC5011 will remove DNSKEY_K_1 from the set of trust anchors. That is okay, since they have already added DNSKEY_K_2 as the new trust anchor. Thus, algorithm 2 is the only signaled algorithm by now. That means, we only need RRSIG_K_2(DNSKEY) to authenticate the DNSKEY RRset, and we still are compliant with section 2.2 from RFC 4035: There must be a RRSIG for each RRset using at least one DNSKEY of each algorithm in the zone apex DNSKEY RRset.

The lesson of all of this is that a Single Type Signing scheme algorithm rollover using RFC5011 is as complicated as the name of the rollover implies, one is better off explicitly using a split key temporarily.

Also see Figure 12 in Appendix C.

4.1.6. Considerations for Automated Key Rollovers

As keys must be renewed periodically, there is some motivation to automate the rollover process. Consider the following:

- o ZSK rollovers are easy to automate as only the child zone is involved.
- o A KSK rollover needs interaction between parent and child. Data exchange is needed to provide the new keys to the parent; consequently, this data must be authenticated and integrity must be guaranteed in order to avoid attacks on the rollover.

4.2. Planning for Emergency Key Rollover

This section deals with preparation for a possible key compromise. Our advice is to have a documented procedure ready for when a key compromise is suspected or confirmed.

When the private material of one of your keys is compromised it can be used for as long as a valid trust chain exists. A trust chain remains intact for

- o as long as a signature over the compromised key in the trust chain is valid,
- o as long as the DS RR in the parent zone points to the compromised key,
- o as long as the key is anchored in a resolver and is used as a starting point for validation (this is generally the hardest to update).

While a trust chain to your compromised key exists, your namespace is vulnerable to abuse by anyone who has obtained illegitimate possession of the key. Zone operators have to make a trade-off if the abuse of the compromised key is worse than having data in caches that cannot be validated. If the zone operator chooses to break the trust chain to the compromised key, data in caches signed with this key cannot be validated. However, if the zone administrator chooses to take the path of a regular rollover, during the rollover the malicious key holder can continue to spoof data so that it appears to be valid.

4.2.1. KSK Compromise

A zone containing a DNSKEY RRSset with a compromised KSK is vulnerable as long as the compromised KSK is configured as trust anchor or a DS record in the parent zone points to it.

A compromised KSK can be used to sign the key set of an attacker's zone. That zone could be used to poison the DNS.

Therefore, when the KSK has been compromised, the trust anchor or the parent DS record should be replaced as soon as possible. It is local policy whether to break the trust chain during the emergency rollover. The trust chain would be broken when the compromised KSK is removed from the child's zone while the parent still has a DS record pointing to the compromised KSK (the assumption is that there is only one DS record at the parent. If there are multiple DS records this does not apply -- however the chain of trust of this

particular key is broken).

Note that an attacker's zone still uses the compromised KSK and the presence of the corresponding DS record in the parent would cause the data in this zone to appear as valid. Removing the compromised key would cause the attacker's zone to appear as valid and the child's zone as Bogus. Therefore, we advise not to remove the KSK before the parent has a DS record for the new KSK in place.

4.2.1.1. Keeping the Chain of Trust Intact

If we follow this advice, the timing of the replacement of the KSK is somewhat critical. The goal is to remove the compromised KSK as soon as the new DS RR is available at the parent. We therefore have to make sure that the signature made with a new KSK over the key set that contains the compromised KSK expires just after the new DS appears at the parent. Expiration of that signature will cause expiration of that key set from the caches.

The procedure is as follows:

1. Introduce a new KSK into the key set, keep the compromised KSK in the key set.
2. Sign the key set, with a short validity period. The validity period should expire shortly after the DS is expected to appear in the parent and the old DSes have expired from caches.
3. Upload the DS for this new key to the parent.
4. Follow the procedure of the regular KSK rollover: Wait for the DS to appear in the authoritative servers and then wait as long as the TTL of the old DS RRs. If necessary re-sign the DNSKEY RRSet and modify/extend the expiration time.
5. Remove the compromised DNSKEY RR from the zone and re-sign the key set using your "normal" validity interval.

An additional danger of a key compromise is that the compromised key could be used to facilitate a legitimate DNSKEY/DS rollover and/or nameserver changes at the parent. When that happens, the domain may be in dispute. An authenticated out-of-band and secure notify mechanism to contact a parent is needed in this case.

Note that this is only a problem when the DNSKEY and or DS records are used for authentication at the parent.

4.2.1.2. Breaking the Chain of Trust

There are two methods to break the chain of trust. The first method causes the child zone to appear 'Bogus' to validating resolvers. The other causes the child zone to appear 'insecure'. These are described below.

In the method that causes the child zone to appear 'Bogus' to validating resolvers, the child zone replaces the current KSK with a new one and re-signs the key set. Next it sends the DS of the new key to the parent. Only after the parent has placed the new DS in the zone is the child's chain of trust repaired.

An alternative method of breaking the chain of trust is by removing the DS RRs from the parent zone altogether. As a result, the child zone would become insecure.

4.2.2. ZSK Compromise

Primarily because there is no interaction with the parent required when a ZSK is compromised, the situation is less severe than with a KSK compromise. The zone must still be re-signed with a new ZSK as soon as possible. As this is a local operation and requires no communication between the parent and child, this can be achieved fairly quickly. However, one has to take into account that just as with a normal rollover the immediate disappearance of the old compromised key may lead to verification problems. Also note that until the RRSIG over the compromised ZSK has expired, the zone may be still at risk.

4.2.3. Compromises of Keys Anchored in Resolvers

A key can also be pre-configured in resolvers. For instance, if DNSSEC is successfully deployed the root key may be pre-configured in most security aware resolvers.

If trust-anchor keys are compromised, the administrators of resolvers using these keys should be notified of this fact. Zone administrators may consider setting up a mailing list to communicate the fact that a SEP key is about to be rolled over. This communication will of course need to be authenticated by some means, e.g. by using digital signatures.

End-users faced with the task of updating an anchored key should always validate the new key. New keys should be authenticated out-of-band, for example, through the use of an announcement website that is secured using secure sockets (TLS) [22].

4.3. Parent Policies

4.3.1. Initial Key Exchanges and Parental Policies Considerations

The initial key exchange is always subject to the policies set by the parent. It is specifically important in a registry-registrar model where the key material is to be passed from the DNS operator, to the (parent) registry via a registrar, where both DNS operator and registrar are selected by the registrant and might be different organisations. When designing a key exchange policy one should take into account that the authentication and authorization mechanisms used during a key exchange should be as strong as the authentication and authorization mechanisms used for the exchange of delegation information between parent and child. That is, there is no implicit need in DNSSEC to make the authentication process stronger than it is for regular DNS.

Using the DNS itself as the source for the actual DNSKEY material, with an out-of-band check on the validity of the DNSKEY, has the benefit that it reduces the chances of user error. A DNSKEY query tool can make use of the SEP bit [5] to select the proper key from a DNSSEC key set, thereby reducing the chance that the wrong DNSKEY is sent. It can validate the self-signature over a key; thereby verifying the ownership of the private key material. Fetching the DNSKEY from the DNS ensures that the chain of trust remains intact once the parent publishes the DS RR indicating the child is secure.

Note: the out-of-band verification is still needed when the key material is fetched via the DNS. The parent can never be sure whether or not the DNSKEY RRs have been spoofed.

4.3.2. Storing Keys or Hashes?

When designing a registry system one should consider which of the DNSKEYs and/or the corresponding DSes to store. Since a child zone might wish to have a DS published using a message digest algorithm not yet understood by the registry, the registry can't count on being able to generate the DS record from a raw DNSKEY. Thus, we recommend that registry systems at least support storing DS records (also see draft-ietf-dnsop-dnssec-trut-anchor [26]).

It may also be useful to store DNSKEYs, since having them may help during troubleshooting and, as long as the child's chosen message digest is supported, the overhead of generating DS records from them is minimal. Having an out-of-band mechanism, such as a registry directory (e.g., Whois), to find out which keys are used to generate DS Resource Records for specific owners and/or zones may also help with troubleshooting.

The storage considerations also relate to the design of the customer interface and the method by which data is transferred between registrant and registry; Will the child zone administrator be able to upload DS RRs with unknown hash algorithms or does the interface only allow DNSKEYs? When Registries support the Extensible Provisioning Protocol (EPP) [17], that can be used for registrar-registry interactions since that protocol allows the transfer of both DS and optionally DNSKEY RRs. There is no standardized way for moving the data between the customer and the registrar. Different registrars have different mechanisms, ranging from simple web interfaces to various APIs. In some cases the use of the DNSSEC extensions to EPP may be applicable.

4.3.3. Security Lameness

Security lameness is defined as the state whereby the parent has a DS RR pointing to a non-existing DNSKEY RR. Security lameness may occur temporarily during a double-DS rollover scheme. However care should be taken that not all DS RRs are security lame which may cause the child's zone to be marked "Bogus" by verifying DNS clients.

As part of a comprehensive delegation check, the parent could, at key exchange time, verify that the child's key is actually configured in the DNS. However, if a parent does not understand the hashing algorithm used by child, the parental checks are limited to only comparing the key id.

Child zones should be very careful in removing DNSKEY material, specifically SEP keys, for which a DS RR exists.

Once a zone is "security lame", a fix (e.g., removing a DS RR) will take time to propagate through the DNS.

4.3.4. DS Signature Validity Period

Since the DS can be replayed as long as it has a valid signature, a short signature validity period for the DS RRSIG minimizes the time a child is vulnerable in the case of a compromise of the child's KSK(s). A signature validity period that is too short introduces the possibility that a zone is marked "Bogus" in case of a configuration error in the signer. There may not be enough time to fix the problems before signatures expire (this is a generic argument also see Section 4.4.2). Something as mundane as operator unavailability during weekends shows the need for DS signature validity periods longer than two days. We recommend an absolute minimum for a DS signature validity period of a few days.

The maximum signature validity period of the DS record depends on how

long child zones are willing to be vulnerable after a key compromise. On the other hand, shortening the DS signature validity interval increases the operational risk for the parent. Therefore, the parent may have policy to use a signature validity interval that is considerably longer than the child would hope for.

A compromise between the operational constraints of the parent and minimizing damage for the child may result in a DS signature validity period somewhere between a week and months.

In addition to the signature validity period, which sets a lower bound on the number of times the zone owner will need to sign the zone data and which sets an upper bound to the time a child is vulnerable after key compromise, there is the TTL value on the DS RRs. Shortening the TTL means that the authoritative servers will see more queries. But on the other hand, a short TTL lowers the persistence of DS RRsets in caches thereby increasing the speed with which updated DS RRsets propagate through the DNS.

4.3.5. Changing DNS Operators

The parent-child relation is often described in terms of a (thin) registry model. Where a registry maintains the parent zone, and the registrant (the user of the child-domain name), deals with the registry through an intermediary called a registrar. (See [11] for a comprehensive definition). Registrants may out-source the maintenance of their DNS system, including the maintenance of DNSSEC key material, to the registrar or to another third party, which we will call the DNS operator. The DNS operator that has control over the DNS zone and its keys may prevent the registrant to make a timely move to a different DNS operator.

For various reasons, a registrant may want to move between DNS operators. How easy this move will be depends principally on the DNS operator from which the registrant is moving (the losing operator), as they have control over the DNS zone and its keys. The following sections describe the two cases: where the losing operator cooperates with the new operator (the gaining operator), and where the two do not cooperate.

4.3.5.1. Cooperationg DNS operators

In this scenario, it is assumed that losing operator will not pass any private key material to the gaining operator (that would constitute a trivial case) but is otherwise fully cooperative.

In this environment one could proceed with a pre-publish ZSK rollover whereby the losing operator pre-publishes the ZSK of the gaining

operator, combined with a double signature KSK rollover where the two registrars exchange public KSKs and independently generate a signature over those keysets that they combine and both publish in their copy of the zone. Once that is done they can use their own private keys to sign any of their zone content during the transfer.

| ----- | | |
|-------------------|-------------------|-------------------|
| initial | | pre-publish |
| ----- | | |
| Parent: | | |
| NS_A | | NS_A |
| DS_A | | DS_A |
| ----- | | |
| Child at A: | Child at A: | Child at B: |
| SOA_A0 | SOA_A1 | SOA_B0 |
| RRSIG_Z_A(SOA) | RRSIG_Z_A(SOA) | RRSIG_Z_B(SOA) |
| NS_A | NS_A | NS_B |
| RRSIG_Z_A(NS) | NS_B | RRSIG_Z_B(NS) |
| | RRSIG_Z_A(NS) | |
| | | |
| DNSKEY_Z_A | DNSKEY_Z_A | DNSKEY_Z_A |
| DNSKEY_K_A | DNSKEY_Z_B | DNSKEY_K_B |
| RRSIG_Z_A(DNSKEY) | DNSKEY_K_A | DNSKEY_K_A |
| RRSIG_K_A(DNSKEY) | DNSKEY_K_B | DNSKEY_K_B |
| | RRSIG_Z_B(DNSKEY) | RRSIG_Z_B(DNSKEY) |
| | RRSIG_K_B(DNSKEY) | RRSIG_K_B(DNSKEY) |
| | RRSIG_Z_A(DNSKEY) | RRSIG_Z_A(DNSKEY) |
| | RRSIG_K_A(DNSKEY) | RRSIG_K_A(DNSKEY) |
| ----- | | |

| ----- | | |
|----------------|----------------|----------------|
| Redelegation | | post migration |
| ----- | | |
| Parent: | | |
| | NS_B | NS_B |
| | DS_B | DS_B |
| ----- | | |
| Child at A: | Child at B: | Child at B: |
| SOA_A2 | SOA_B1 | SOA_B2 |
| RRSIG_Z_A(SOA) | RRSIG_Z_B(SOA) | RRSIG_Z_B(SOA) |
| NS_A | NS_B | NS_B |
| NS_B | RRSIG_Z_B(NS) | RRSIG_Z_B(NS) |

RRSIG_Z_A(NS)

| | | |
|-------------------|-------------------|-------------------|
| DNSKEY_Z_A | DNSKEY_Z_A | DNSKEY_Z_B |
| DNSKEY_Z_B | DNSKEY_Z_B | DNSKEY_K_B |
| DNSKEY_K_A | DNSKEY_K_A | RRSIG_Z_B(DNSKEY) |
| DNSKEY_K_B | DNSKEY_K_B | RRSIG_K_B(DNSKEY) |
| RRSIG_Z_B(DNSKEY) | RRSIG_Z_B(DNSKEY) | |
| RRSIG_K_B(DNSKEY) | RRSIG_K_B(DNSKEY) | |
| RRSIG_Z_A(DNSKEY) | RRSIG_Z_A(DNSKEY) | |
| RRSIG_K_A(DNSKEY) | RRSIG_K_A(DNSKEY) | |

Figure 9: Rollover for cooperating operators

In this figure A denotes the losing operator and B the gaining operator. RRSIGZ is the RRSIG produced by a ZSK, RRSIGK is produced with a KSK, the appended A or B indicates the producers of the key pair. Child at A is how the zone content is represented by the losing DNS operator and Child at B is how the zone content is represented by the gaining DNS operator.

4.3.5.2. Non Cooperationg DNS Operators

In the non-cooperative case matters are more complicated. The losing operator may not cooperate and leave the data in the DNS as is. In the extreme case the losing operator may become obstructive and publish a DNSKEY RR with a high TTL and corresponding signature validity so that registrar A's DNSKEY could end up in caches for (in theory at least) tens of years.

The problem arises when a validator tries to validate with the losing operator's key and there is no signature material produced with the losing operator available in the delegation path after redelegation from the loosing operator to the gaining operator has taken place. One could imagine a rollover scenario where the gaining operator pulls all RRSIGs created by the losing operator and publishes those in conjunction with its own signatures, but that would not allow any changes in the zone content. Since a redelegation took place the NS RRset has - by definition - changed so such rollover scenario will not work. Besides if zone transfers are not allowed by the losing operator and NSEC3 is deployed in the losing operator's zone, then the gaining operator's zone will not have certainty that all of A's RRSIGs are transferred.

The only viable option for the registrant is to publish its zone unsigned and ask the registry to remove the DS RR pointing to the

losing operator's DNSKEY for as long as the DNSKEY of the losing operator, or any of the signatures produced by it are likely to disappear in caches, which as mentioned above could in theory be for tens of years.

Note that implementations limit the time DNSKEYs that seem to be unable to validate signatures are cached and/or will try to recover from cases where DNSKEYs do not seem to be able to validate data. Although that is not a protocol requirement it seems that that practice may limit the impact of this problem the problem of non-cooperating registrars.

However, there is no operational methodology to work around this business issue, and proper contractual relationships between all involved parties seems to be the only solution to cope with these problems. It should be noted that in many cases, the problem with temporary broken delegations already exists when a zone changes from one DNS operator to another. Besides, it is often the case that when operators are changed the services that that zone references also change operator, possibly involving some downtime.

In any case, to minimise such problems, the classic recommendation is to have relative short TTL on all involved resource records. That will solve many of the problems regarding changes to a zone regardless of whether DNSSEC is used.

4.4. Time in DNSSEC

Without DNSSEC, all times in the DNS are relative. The SOA fields REFRESH, RETRY, and EXPIRATION are timers used to determine the time elapsed after a slave server synchronized with a master server. The Time to Live (TTL) value and the SOA RR minimum TTL parameter [9] are used to determine how long a forwarder should cache data after it has been fetched from an authoritative server. By using a signature validity period, DNSSEC introduces the notion of an absolute time in the DNS. Signatures in DNSSEC have an expiration date after which the signature is marked as invalid and the signed data is to be considered Bogus.

The considerations in this section are all qualitative and focused on the operational and managerial issues. A more thorough quantitative analysis of rollover timing parameters can be found in draft-ietf-dnsop-dnssec-key-timing [24]

4.4.1. Time Considerations

Because of the expiration of signatures, one should consider the following:

- o We suggest the Maximum Zone TTL of your zone data to be a fraction of your signature validity period.

If the TTL was of similar order as the signature validity period, then all RRSets fetched during the validity period would be cached until the signature expiration time. Section 8.1 of RFC4033 [3] suggests that "the resolver may use the time remaining before expiration of the signature validity period of a signed RRSset as an upper bound for the TTL". As a result, query load on authoritative servers would peak at signature expiration time, as this is also the time at which records simultaneously expire from caches.

To avoid query load peaks, we suggest the TTL on all the RRs in your zone to be at least a few times smaller than your signature validity period.

- o We suggest the signature publication period to end at least one Maximum Zone TTL duration before the end of the signature validity period.

Re-signing a zone shortly before the end of the signature validity period may cause simultaneous expiration of data from caches. This in turn may lead to peaks in the load on authoritative servers. To avoid this schemes are deployed whereby the zone is periodically visited for a resigning operation and those signatures that are within a so called refresh interval from signature expiration are recreated. Also see Section 4.4.2 below.

- o We suggest the Minimum Zone TTL to be long enough to both fetch and verify all the RRs in the trust chain. In workshop environments, it has been demonstrated [18] that a low TTL (under 5 to 10 minutes) caused disruptions because of the following two problems:

1. During validation, some data may expire before the validation is complete. The validator should be able to keep all data until it is completed. This applies to all RRs needed to complete the chain of trust: DS, DNSKEY, RRSIG, and the final answers, i.e., the RRSset that is returned for the initial query.

2. Frequent verification causes load on recursive nameservers. Data at delegation points, DS, DNSKEY, and RRSIG RRs benefit from caching. The TTL on those should be relatively long.

- o Slave servers will need to be able to fetch newly signed zones well before the RRSIGs in the zone served by the slave server pass their signature expiration time.

When a slave server is out of synchronization with its master and data in a zone is signed by expired signatures, it may be better for the slave server not to give out any answer.

Normally, a slave server that is not able to contact a master server for an extended period will expire a zone. When that happens, the server will respond differently to queries for that zone. Some servers issue SERVFAIL, whereas others turn off the 'AA' bit in the answers. The time of expiration is set in the SOA record and is relative to the last successful refresh between the master and the slave servers. There exists no coupling between the signature expiration of RRSIGs in the zone and the expire parameter in the SOA.

If the server serves a DNSSEC zone, then it may well happen that the signatures expire well before the SOA expiration timer counts down to zero. It is not possible to completely prevent this by modifying the SOA parameters.

However, the effects can be minimized where the SOA expiration time is equal to or shorter than the signature validity period.

The consequence of an authoritative server not being able to update a zone for an extended period of time is that signatures may expire. In this case non-secure resolvers will continue to be able to resolve data served by the particular slave servers while security-aware resolvers will experience problems because of answers being marked as Bogus.

We suggest the SOA expiration timer being approximately one third or a quarter of the signature validity period. It will allow problems with transfers from the master server to be noticed before the actual signature times out.

We also suggest that operators of nameservers that supply secondary services develop systems to identify upcoming signature expirations in zones they slave and take appropriate action where such an event is detected.

When determining the value for the expiration parameter one has to take the following into account: what are the chances that all my secondaries expire the zone? How quickly can I reach an administrator of secondary servers to load a valid zone? These questions are not DNSSEC specific but may influence the choice

of your signature validity intervals.

4.4.2. Signature Validation Periods

4.4.2.1. Maximum Value

The first consideration for choosing a maximum signature validity period is the risk of a replay attack. For low-value, long-term stable resources the risks may be minimal and the signature validity period may be several months. Although signature validity periods of many years are allowed the same operational habit arguments as in Section 3.2.2 play a role: when a zone is re-signed with some regularity then operators remain conscious about the operational necessity of re-signing.

4.4.2.2. Minimum Value

The minimum value of the signature validity period is set for the time by which one would like to survive operational failure in provisioning: what is the time that a failure will be noticed, what is the time that action is expected to be taken? By answering these questions availability of operators during (long) weekends or time taken to access to backup media can be taken into account. The result could easily suggest a minimum Signature Validity period of a few days.

Note however, the argument above is assuming that zone data has just been signed and published when the problem occurred. In practice it may be that a zone is signed according to a frequency set by the Re-Sign Period whereby the signer visits the zone content and only refreshes signatures that are close to expiring: the signer will only refresh signatures if they are within the Refresh Period from the signature expiration time. The Re-Sign Period must be smaller than the Refresh Period in order for zone data to be signed in timely fashion.

If an operational problem occurs during resigning then the signatures in the zone to expire first are the ones that have been generated longest ago. In the worst case these signatures are the Refresh Period minus the Re-Sign Period away from signature expiration.

In other words, the minimum Signature Validity interval is set by first choosing the Refresh Period (usually a few days), then defining the Re-Sign period in such a way that the Refresh Period minus the Resign period sets the time in which operational havoc can be resolved.

To make matters slightly more complicated, some signers vary the

signature validity period over a small range (the jitter interval) so that not all signatures expire at the same time. The jitter should not influence your calculation as long as it is smaller than the refresh period and the resign period is at least half the refresh period.

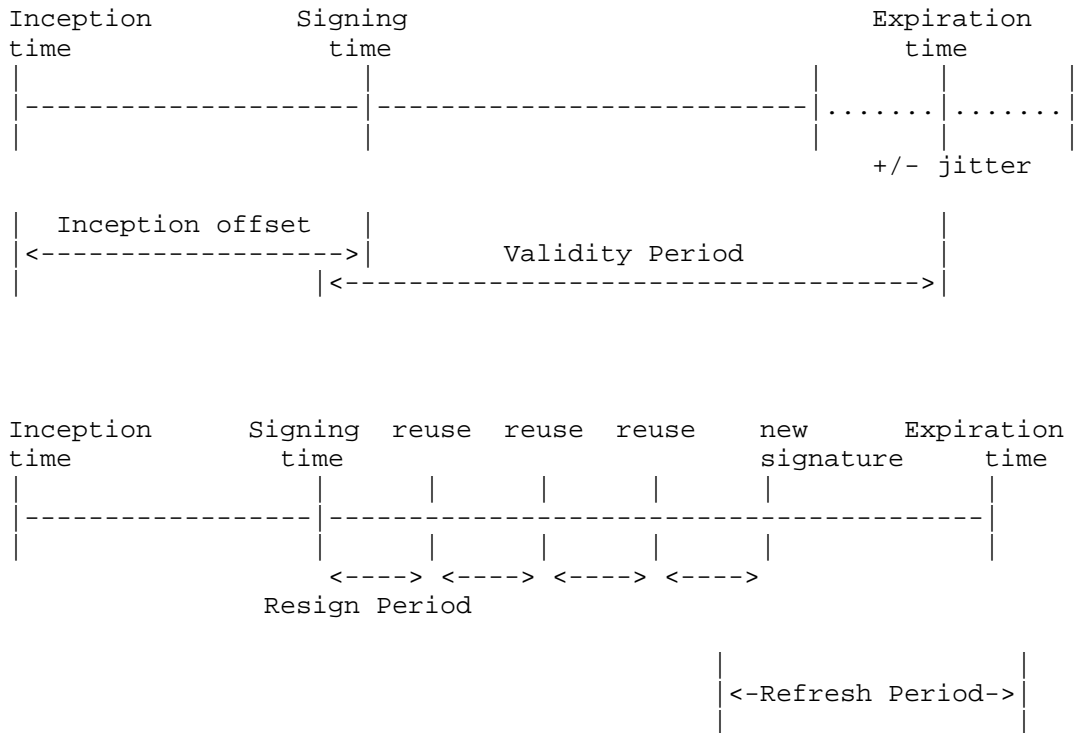


Figure 10: Signature Timing Parameters

Note that in the figure the validity of the signature starts shortly before the inception time. That is done to deal with validators that might have some clock skew.

4.4.2.3. Differentiation between RR sets

It is possible to vary signature validity periods between signatures over different RR sets in the zone. In practice this could be done when zones contain highly volatile data (which may be the case in dynamic update environments). Note however that the risk of replay (e.g. by stale secondary servers) is what should be leading in determining the signature validity period since the TTL on the data itself still are the primary parameter for cache expiry.

In some cases the risk of replaying existing data might be different from the risk of replaying the denial of data. In those cases the signature validity period on NSEC or NSEC3 records may be tweaked accordingly.

When a zone contains secure delegations then a relatively short signature validity interval protects the child against replay attacks, in the case the child's key is compromised (see Section 4.3.4). Since there is a higher operational risk for the parent registry when choosing a short validity interval and a higher operational risk for the child when choosing a long validity period some (price) differentiation may occur for validity periods between individual DS RRs in a single zone.

There seem to be no other arguments for differentiation in validity periods.

4.4.2.4. Other timing parameters in a zone

The arguments for tuning minimum signature validity period are remarkably similar to the arguments used to set the SOA expiration timer. It is advised to set the SOA expiration to a value greater than the signature validity period.

5. Next Record type

One of the design tradeoffs made during the development of DNSSEC was to separate the signing and serving operations instead of performing cryptographic operations as DNS requests are being serviced. It is therefore necessary to create records that cover the very large number of non-existent names that lie between the names that do exist.

There are two mechanisms to provide authenticated proof of non-existence of domain names in DNSSEC: a clear text one and an obfuscated-data one. Each mechanism:

- o includes a list of all the RRTYPEs present which can be used to prove the non-existence of RRTYPEs at a certain name;
- o stores only the name for which the zone is authoritative (that is, glue in the zone is omitted); and
- o uses a specific RRTYPE to store information about the RRTYPEs present at the name: the clear-text mechanism uses NSEC, and the obfuscated-data mechanism uses NSEC3.

5.1. Differences between NSEC and NSEC3

The clear text mechanism (NSEC) is implemented using a sorted linked list of names in the zone. The obfuscated-data mechanism (NSEC3) is similar but first hashes the names using a one-way hash function, before creating a sorted linked list of the resulting (hashed) strings.

The NSEC record requires no cryptographic operations aside from the validation of its associated signature record. It is human readable and can be used in manual queries to determine correct operation. The disadvantage is that it allows for "zone walking", where one can request all the entries of a zone by following the linked list of NSEC RRs via the "Next Domain Name" field.

Though all agree DNS data is accessible through query mechanisms, a side effect of NSEC is that it allows the contents of a zone file to be enumerated in full by sequential queries. Whilst for some operators this behaviour is acceptable or even desirable, for others it is undesirable for policy, regulatory or other reasons. This is the first difference between NSEC and NSEC3.

The second difference between NSEC and NSEC3 is that NSEC requires a signature over every RR in the zonefile, thereby ensuring that any denial of existence is cryptographically signed. However, in a large zonefile containing many delegations very few of which are to signed zones, this may produce unacceptable additional overhead especially where insecure delegations are subject to frequent update (a typical example might be a TLD operator with few registrants using secure delegations). NSEC3 allows intervals between two such delegations to "Opt-out" in which case they may contain one more more insecure delegations, thus reducing the size and cryptographic complexity of the zone at the expense of the ability to cryptographically deny the existence of names in a specific span.

The NSEC3 record uses a hashing method of the requested RRlabel. To increase the workload required to guess entries in the zone, the number of hashing iteration's can be specified in the NSEC3 record. Additionally, a salt can be specified that also modifies the hashes. Note that NSEC3 does not give full protection against information leakage from the zone.

5.2. NSEC or NSEC3

The first motivation to deploy NSEC3, prevention of zone enumeration, only makes sense when zone content is not highly structured or trivially guessable. Highly structured zones such as the in-addr.arpa, ip6.arpa and el64.arpa can be trivially enumerated using

ordinary DNS properties while for small zones that only contain contain records in the APEX and a few common RRlabels such as "www" or "mail" guessing zone content and proving completeness is also trivial when using NSEC3.

In those cases the use of NSEC is recommended to ease the work required by signers and validating resolvers.

For large zones where there is an implication of "not readily available" RRlabels, such as those where one has to sign a non-disclosure agreement before obtaining it, NSEC3 is recommended.

The considerations for the second reason to deploy NSEC3 are discussed below (Section 5.3.4).

5.3. NSEC3 parameters

The NSEC3 hashing algorithm is performed on the Fully Qualified Domain Name (FQDN) in its uncompressed form. This ensures brute force work done by an attacker for one (FQDN) RRlabel cannot be re-used for another (FQDN) RRlabel attack, as these entries are, by definition unique.

5.3.1. NSEC3 Algorithm

At the moment of writing there is only one NSEC3 Hashing algorithm defined. [21] specifically calls out that when a new hash algorithm for use with NSEC3 is specified, a transition mechanism MUST also be defined. Therefore this document does not consider NSEC3 hash algorithm transition.

5.3.2. NSEC3 Iterations

One of the concerns with NSEC3 is a pre-calculated dictionary attack could be made in order to assess if certain domain names exist within the zones or not. Two mechanisms are introduced in the NSEC3 specification to increase the costs of such dictionary attacks: Iterations and Salt.

RFC5155 Section 10.3 [21] considers the trade-offs between incurring cost during the signing process and imposing costs to the validating nameserver, while still providing a reasonable barrier against dictionary attacks. It provides useful limits of iterations for a given RSA key size. These are 150 iterations for 1024 bit keys, 500 iterations for 2048 bit keys and 2,500 iterations for 4096 bit keys. Choosing two-thirds of the maximum is deemed to be a sufficiently costly yet not excessive value.

5.3.3. NSEC3 Salt

While the NSEC3 iterations parameter increases the cost of hashing a dictionary word, the NSEC3 salt reduces the lifetime for which that calculated hash can be used. A change of the salt value by the zone owner would cause an attacker to lose all precalculated work for that zone.

The FQDN RRLabel, which is part of the value that is hashed, already ensures that brute force work for one RRLabel can not be re-used to attack other RRLabel (e.g. in other domains) due to their uniqueness.

The salt of all NSEC3 records in a zone needs to be the same. Since changing the salt requires all the NSEC3 records to be regenerated, and thus requires generating new RRSIG's over these NSEC3 records, it is recommended to align the change of the salt with a change of the Zone Signing Key, as that process in itself already requires all RRSIG's to be regenerated. If there is no critical dependency on incremental signing and the whole zone can be signed with little effort there is no need for such alignment. However, unlike Zone Signing Key changes, NSEC3 salt changes do not need special rollover procedures. It is possible to change the salt each time the zone is updated.

5.3.4. Opt-out

The Opt-Out mechanism was introduced to allow for a gradual introduction of signed records in zones that contain mostly delegation records. The use of the OPT-OUT flag changes the meaning of the NSEC3 span from authoritative denial of the existence of names within the span to a proof that DNSSEC is not available for the delegations within the span. [Editors Note: One could make this construct more correct by talking about the hashed names and the hashed span, but I believe that is overkill]. This allows for the addition or removal of the delegations covered by the span without recalculating or re- signing RRs in the NSEC3 RR chain.

Opt-Out is specified to be used only over delegation points and will therefore only bring relief to zones with a large number of zones and where the number of secure delegations is small. This consideration typically holds for large top-level-domains and similar zones; in most other circumstances Opt-Out should not be deployed. Further considerations can be found in RFC5155 section 12.2 [21].

6. Security Considerations

DNSSEC adds data integrity to the DNS. This document tries to assess the operational considerations to maintain a stable and secure DNSSEC

service. Not taking into account the 'data propagation' properties in the DNS will cause validation failures and may make secured zones unavailable to security-aware resolvers.

7. IANA considerations

There are no IANA considerations with respect to this document

8. Contributors and Acknowledgments

Significant parts of the text of this document is copied from RFC4641 [14]. That document was edited by Olaf Kolkman and Miek Gieben. Other people that contributed or where otherwise involved in that work were in random order: Rip Loomis, Olafur Gudmundsson, Wesley Griffin, Michael Richardson, Scott Rose, Rick van Rein, Tim McGinnis, Gilles Guede, Olivier Courtay, Sam Weiler, Jelte Jansen, Niall O'Reilly, Holger Zuleger, Ed Lewis, Hilarie Orman, Marcos Sanz, Peter Koch, Mike StJohns, Emma Bretherick, Adrian Bedford, and Lindy Foster, and O. Courtay.

For this version of the document we would like to acknowledge a few people for significant contributions:

Paul Hoffman for his contribution on the choice of cryptographic parameters and addressing some of the trust anchor issues;

Jelte Jansen who provided the initial text in Section 4.1.5;

Paul Wouters who provided the initial text for Section 5 and Alex Bligh who improved it;

Erik Rescorla whos blogpost on "the Security of ZSK rollovers" inspired text in Section 3.1;

Stephen Morris who made a pass on English style and grammar;

Matthijs Mekking thouroughly reviewed and provided concrete improvements on the specific types of keyrollovers (e.g. he provided the tables in Appendix C); and

Olafur Gudmundsson and Onrej Sury who provided input on Section 4.1.5 based on actual operational experience.

The figure in Section 4.4.2 was adapted from the OpenDNSSEC user documentation.

In addition valuable contributions in the form of text, comments, or review where provided by Mark Andrews, Patrik Faltstrom, Tony Finch,

Alfred Hines, Bill Manning, Scott Rose, and Wouter Wijngaards.

[EDITOR NOTE: please let me know if there is an oversight here]

9. References

9.1. Normative References

- [1] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.
- [2] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.
- [3] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.
- [4] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.
- [5] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.

9.2. Informative References

- [6] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [7] Ohta, M., "Incremental Zone Transfer in DNS", RFC 1995, August 1996.
- [8] Vixie, P., "A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)", RFC 1996, August 1996.
- [9] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, March 1998.
- [10] Wellington, B., "Secure Domain Name System (DNS) Dynamic Update", RFC 3007, November 2000.
- [11] Hollenbeck, S., "Generic Registry-Registrar Protocol Requirements", RFC 3375, September 2002.
- [12] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", BCP 86, RFC 3766,

April 2004.

- [13] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.
- [14] Kolkman, O. and R. Gieben, "DNSSEC Operational Practices", RFC 4641, September 2006.
- [15] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.
- [16] StJohns, M., "Automated Updates of DNS Security (DNSSEC) Trust Anchors", RFC 5011, September 2007.
- [17] Gould, J. and S. Hollenbeck, "Domain Name System (DNS) Security Extensions Mapping for the Extensible Provisioning Protocol (EPP)", RFC 5910, May 2010.
- [18] Rose, S., "NIST DNSSEC workshop notes", , June 2001.
- [19] Barker, E. and J. Kelsey, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)", Nist Special Publication 800-90, March 2007.
- [20] Hardaker, W., "Use of SHA-256 in DNSSEC Delegation Signer (DS) Resource Records (RRs)", RFC 4509, May 2006.
- [21] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, March 2008.
- [22] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [23] Jansen, J., "Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC", RFC 5702, October 2009.
- [24] Morris, S., Ihren, J., and J. Dickinson, "DNSSEC Key Timing Considerations", draft-ietf-dnsop-dnssec-key-timing-00 (work in progress), July 2010.
- [25] Ljunggren, F., Eklund-Lowinder, A., and T. Okubo, "DNSSEC Policy & Practice Statement Framework", draft-ietf-dnsop-dnssec-dps-framework-02 (work in progress), July 2010.
- [26] Larson, M. and O. Gudmundsson, "DNSSEC Trust Anchor Configuration and Maintenance",

draft-ietf-dnsop-dnssec-trust-anchor-03 (work in progress),
March 2009.

Appendix A. Terminology

In this document, there is some jargon used that is defined in other documents. In most cases, we have not copied the text from the documents defining the terms but have given a more elaborate explanation of the meaning. Note that these explanations should not be seen as authoritative.

Anchored key: A DNSKEY configured in resolvers around the globe. This key is hard to update, hence the term anchored.

Bogus: Also see Section 5 of RFC4033 [3]. An RRSset in DNSSEC is marked "Bogus" when a signature of an RRSset does not validate against a DNSKEY.

Key Signing Key or KSK: A Key Signing Key (KSK) is a key that is used exclusively for signing the apex key set. The fact that a key is a KSK is only relevant to the signing tool.

Key size: The term 'key size' can be substituted by 'modulus size' throughout the document. It is mathematically more correct to use modulus size, but as this is a document directed at operators we feel more at ease with the term key size.

Private and public keys: DNSSEC secures the DNS through the use of public key cryptography. Public key cryptography is based on the existence of two (mathematically related) keys, a public key and a private key. The public keys are published in the DNS by use of the DNSKEY Resource Record (DNSKEY RR). Private keys should remain private.

Key rollover: A key rollover (also called key supercession in some environments) is the act of replacing one key pair with another at the end of a key effectivity period.

Refresh Period: The time at the end of the Signature Validity Period during which signatures are refreshed.

Re-Signing frequency: Frequency with which a signing pass on the zone is performed. Alternatively expressed as "Re-Signing Period". It defines when the zone is exposed to the signer. During a signing pass not all signatures in the zone may be refreshed, that depend refresh frequency/interval.

Secure Entry Point (SEP) key: A KSK that has a DS record in the parent zone pointing to it or is configured as a trust anchor. Although not required by the protocol, we recommend that the SEP flag [5] is set on these keys.

Self-signature: This only applies to signatures over DNSKEYs; a signature made with DNSKEY x, over DNSKEY x is called a self-signature. Note: without further information, self-signatures convey no trust. They are useful to check the authenticity of the DNSKEY, i.e., they can be used as a hash.

Signing Jitter: Jitter applied to the signature validity interval.

Signer: The system that has access to the private key material and signs the Resource Record sets in a zone. A signer may be configured to sign only parts of the zone, e.g., only those RRSets for which existing signatures are about to expire.

Single Type Signing Scheme: A signing scheme whereby the distinction between Zone Signing Keys and Key Signing Keys is not made.

Zone Signing Key (ZSK): A key that is used for signing all data in a zone (except, perhaps, the DNSKEY RRSets). The fact that a key is a ZSK is only relevant to the signing tool.

Singing the zone file: The term used for the event where an administrator joyfully signs its zone file while producing melodic sound patterns.

Zone administrator: The 'role' that is responsible for signing a zone and publishing it on the primary authoritative server.

Appendix B. Typographic Conventions

The following typographic conventions are used in this document:

Key notation: A key is denoted by DNSKEY_x_y, where y is an identifier for the type of key: K for Keys Signing Key, Z for Zone Signing Key and S when there is no distinction made between KSK and ZSKs but the key is used as a secure entry point. The 'x' denotes a number or an identifier, x could be thought of as the key id.

RRSet notations: RRs are only denoted by the type. All other information -- owner, class, rdata, and TTL -- is left out. Thus: "example.com 3600 IN A 192.0.2.1" is reduced to "A". RRSets are a list of RRs. An example of this would be "A1, A2", specifying the RRSet containing two "A" records. This could again be abbreviated

to just "A".

Signature notation: Signatures are denoted as RRSIG_x_y(RRSet), which means that RRSet is signed with DNSKEY_x_y.

Zone representation: Using the above notation we have simplified the representation of a signed zone by leaving out all unnecessary details such as the names and by representing all data by "SOAx"

SOA representation: SOAs are represented as SOAx, where x is the serial number.

RRsets ignored: If the signature of non DNSKEY RRsets have the same parameters as the SOA than those are not mentioned. e.g. In the example below the SOA is signed with the same parameters as the foo.example.com A RRset and the latter is therefore ignored in the abbreviated notation.

Using this notation the following signed zone:

```
example.com. 3600 IN SOA ns1.example.com. olaf.example.net. (
    2005092303 ; serial
    450        ; refresh (7 minutes 30 seconds)
    600        ; retry (10 minutes)
    345600     ; expire (4 days)
    300        ; minimum (5 minutes)
)
3600 RRSIG SOA 5 2 3600 20120824013000 (
    20100424013000 14 example.com.
    NMafnzmmZ8wevpCOI+/JxqWBzPxrnzPnSXfo
    ...
    OMY3rTMA2qorupQXjQ== )
3600 NS ns1.example.com.
3600 NS ns2.example.com.
3600 NS ns3.example.com.
3600 RRSIG NS 5 2 3600 20120824013000 (
    20100424013000 14 example.com.
    p0Cj3wzGoPFftFZjj3jeKGK6wGWLwY6mCBEz
    ...
    +SqZIoVHpve7YBeH46wuyF8w4XknA4Oeimc4
    zAgaJM/MeG08KpeHhg== )
3600 TXT "Net::DNS domain"
3600 RRSIG TXT 5 2 3600 20120824013000 (
    20100424013000 14 example.com.
    o7eP8LISK2TEutFQRvK/+U3wq7t4X+PQaQkp
    ...
    BcQlo99vwn+IS4+Jlg== )
300 NSEC foo.example.com. NS SOA TXT RRSIG NSEC DNSKEY
```

```

300      RRSIG      NSEC 5 2 300 20120824013000 (
                    20100424013000 14 example.com.
                    JtHm8ta0diCWYGu/TdrE10lsYSHblN2i/IX+
                    ...
                    PkXNI/Vgf4t3xZaIyw== )
3600     DNSKEY    256 3 5 (
                    AQPaoHW/nC0fj9HuCW3hACSGiP0AkPS3dQFX
                    ...
                    sAuryjQ/HFa5r4mrbhkJ
                    ) ; key id = 14
3600     DNSKEY    257 3 5 (
                    AQPuiszMMAi36agx/V+7Tw95l8PYmoVjHWvO
                    ...
                    oy88Nh+u2c9HF1tw0naH
                    ) ; key id = 15
3600     RRSIG      DNSKEY 5 2 3600 20120824013000 (
                    20100424013000 14 example.com.
                    HWj/VEr6p/FiUUiL70QQWtk+NBiIsJ9mdj5U
                    ...
                    QhhmMwV3tIxJk2eDRQ== )
3600     RRSIG      DNSKEY 5 2 3600 20120824013000 (
                    20100424013000 15 example.com.
                    P47CUy/xPV8qIEuua4tMKG6ei3LQ8RYv3TwE
                    ...
                    JWL70YiUnUG3m9OL9w== )
foo.example.com. 3600 IN A 192.0.2.2
3600     RRSIG      A 5 3 3600 20120824013000 (
                    20100424013000 14 example.com.
                    xHr023P79YrSHHmtSL0alnlfUt4ywn/vWqsO
                    ...
                    JPV/SA4BkoFxiCPrDQ== )
300      NSEC      example.com. A RRSIG NSEC
300      RRSIG      NSEC 5 3 300 20120824013000 (
                    20100424013000 14 example.com.
                    Aaa4kgKhqY7Lzjq3rlPlFidymOeBEK1T6vUF
                    ...
                    Qe000JyzObxx27pY8A== )

```

is reduced to the following representation:

```

SOA2005092303
RRSIG_Z_14(SOA2005092303)
DNSKEY_K_14
DNSKEY_Z_15
RRSIG_K_14(DNSKEY)
RRSIG_Z_15(DNSKEY)

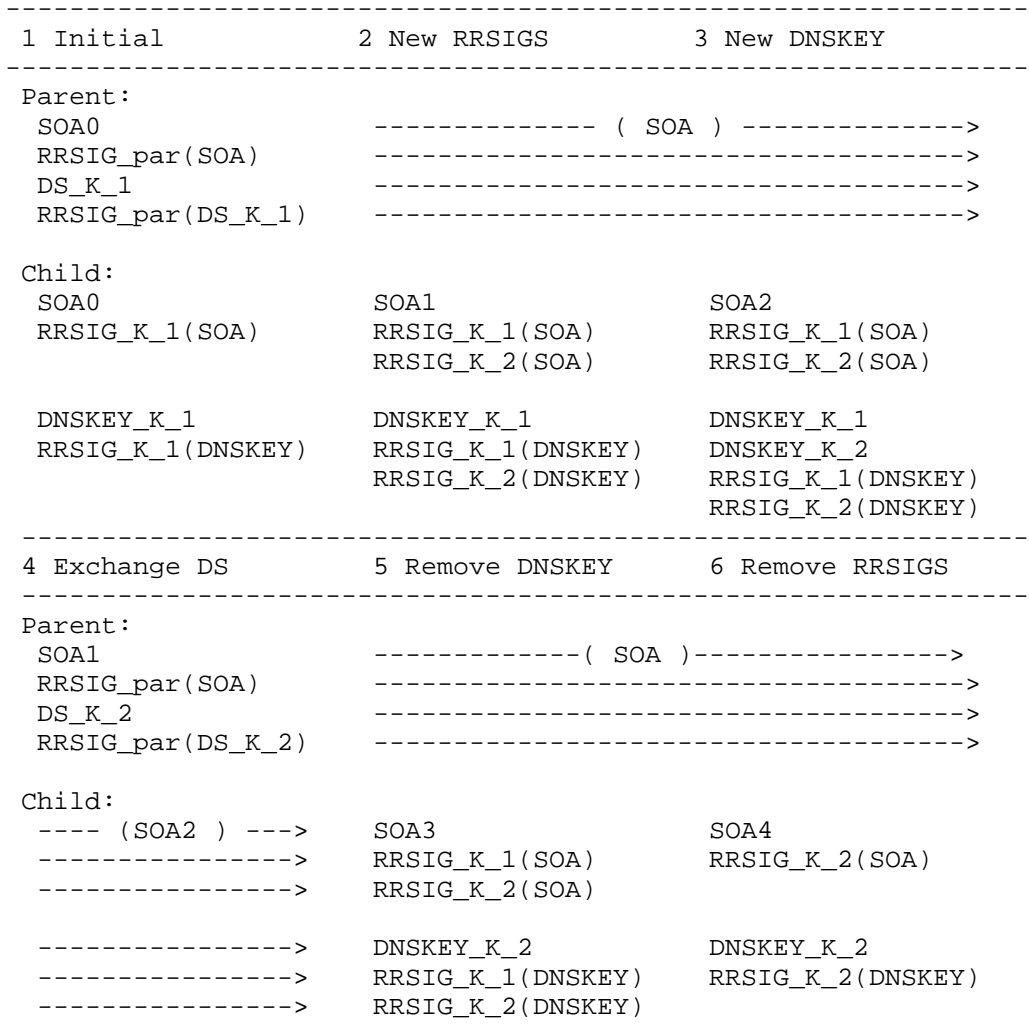
```

The rest of the zone data has the same signature as the SOA record,

i.e., an RRSIG created with DNSKEY 14.

Appendix C. Transition Figures for Special Case Algorithm Rollovers

The figures appendix complement and illustrate the special cases of algorithm rollovers as described in Section 4.1.5



Also see Section 4.1.5.2.

Figure 11: Single Type Signing Scheme Algorithm Roll

```

-----
1 Initial                              2 New RRSIGS                              3 New DNSKEY
-----

Parent:
  SOA0                              ----- ( SOA ) ----->
  RRSIG_par(SOA)                              ----->
  DS_K_1                              ----->
  RRSIG_par(DS_K_1)                              ----->

Child:
  SOA0                              SOA1                              SOA2
  RRSIG_Z_1(SOA)                              RRSIG_Z_1(SOA)                              RRSIG_Z_1(SOA)
  RRSIG_Z_2(SOA)                              RRSIG_Z_2(SOA)                              RRSIG_Z_2(SOA)

  DNSKEY_K_1                              DNSKEY_K_1                              DNSKEY_K_1
  DNSKEY_Z_1                              DNSKEY_Z_1                              DNSKEY_Z_1
  RRSIG_K_1(DNSKEY)                              RRSIG_K_1(DNSKEY)                              DNSKEY_K_2
  RRSIG_K_2(DNSKEY)                              RRSIG_K_2(DNSKEY)                              DNSKEY_Z_2
  RRSIG_K_1(DNSKEY)                              RRSIG_K_2(DNSKEY)

-----
4 Exchange DS                              4b Revoke DNSKEY                              5 Remove DNSKEY
-----

Parent:
  SOA1                              -----( SOA )----->
  RRSIG_par(SOA)                              ----->
  DS_K_2                              ----->
  RRSIG_par(DS_K_2)                              ----->

Child:
  ---- (SOA2 ) --->                              SOA3                              SOA4
  ----->                              RRSIG_Z_1(SOA)                              RRSIG_Z_1(SOA)
  ----->                              RRSIG_Z_2(SOA)                              RRSIG_Z_2(SOA)

  ----->                              DNSKEY_K_1_REVOKED                              DNSKEY_K_2
  ----->                              DNSKEY_Z_1                              DNSKEY_Z_2
  ----->                              DNSKEY_K_2                              RRSIG_K_1_REVOKED(DNSKEY)
  ----->                              DNSKEY_Z_2                              RRSIG_K_2(DNSKEY)
  ----->                              RRSIG_K_1_REVOKED(DNSKEY)
  ----->                              RRSIG_K_2(DNSKEY)

-----
6 Remove RRSIGS
-----

Parent:
  -----( SOA )----->
  ----->
  ----->
  ----->
  ----->

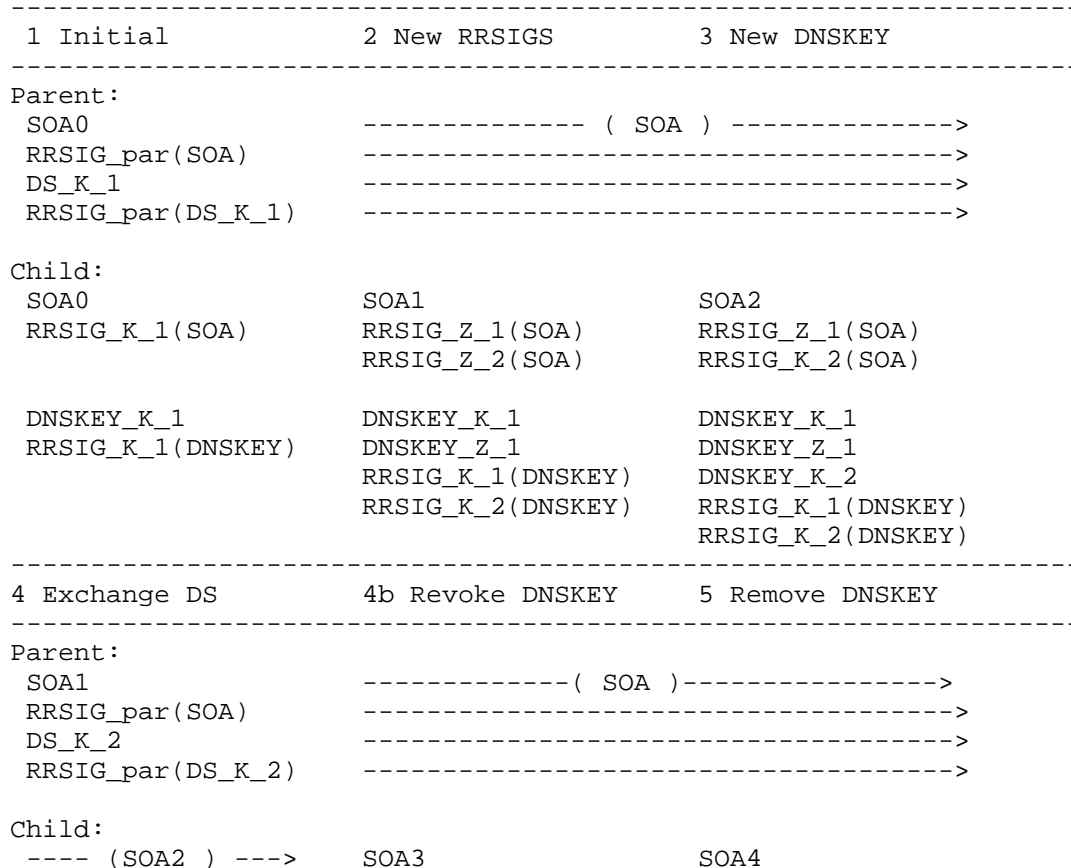
```

```
Child:
  SOA5
  RRSIG_Z_2(SOA)

  DNSKEY_K_2
  DNSKEY_Z_2
  RRSIG_K_2(DNSKEY)
```

Also see Section 4.1.5.3.

Figure 12: RFC5011 Style algorithm roll



```

----->     RRSIG_Z_1(SOA)             RRSIG_Z_1(SOA)
----->     RRSIG_Z_2(SOA)             RRSIG_Z_2(SOA)

----->     DNSKEY_K_1_REVOKED     DNSKEY_K_2
----->     DNSKEY_Z_1                 RRSIG_K_1_REVOKED(DNSKEY)
----->     DNSKEY_K_2                 RRSIG_K_2(DNSKEY)
----->     RRSIG_K_1_REVOKED(DNSKEY)
----->     RRSIG_K_2(DNSKEY)
-----
6 Remove RRSIGS
-----
Parent:
----- ( SOA ) ----->
----->
----->
----->

Child:
SOA5
RRSIG_K_2(SOA)

DNSKEY_K_2
RRSIG_K_2(DNSKEY)
-----

```

Also see Section 4.1.5.4.

Figure 13: RFC5011 algorithm roll in a Single Type Signing Scheme Environment

Appendix D. Document Editing History

[To be removed prior to publication as an RFC]

D.1. draft-ietf-dnsop-rfc4641-00

Version 0 was differs from RFC4641 in the following ways.

- o Status of this memo appropriate for I-D
- o TOC formatting differs.
- o Whitespaces, linebreaks, and pagebreaks may be slightly different because of xml2rfc generation.
- o References slightly reordered.

- o Applied the errata from http://www.rfc-editor.org/errata_search.php?rfc=4641

- o Inserted trivial "IANA considertations" section.

In other words it should not contain substantive changes in content as intended by the workinggroup for the original RFC4641.

D.2. version 0->1

Cryptography details rewritten. (See http://www.nlnetlabs.nl/svn/rfc4641bis/trunk/open-issues/cryptography_flawed)

- o Reference to NIST 800-90 added
- o RSA/SHA256 is being recommended in addition to RSA/SHA1.
- o Complete rewrite of Section 3.4.2 removing the table and suggesting a keysize of 1024 for keys in use for less than 8 years, issued up to at least 2015.
- o Replaced the reference to Schneiers' applied cryptograpy with a reference to RFC4949.
- o Removed the KSK for high level zones consideration

Applied some differentiation with respect of the use of a KSK for parent or trust-anchor relation http://www.nlnetlabs.nl/svn/rfc4641bis/trunk/open-issues/differentiation_trustanchor_parent

http://www.nlnetlabs.nl/svn/rfc4641bis/trunk/open-issues/rollover_assumptions

Added Section 4.1.5 as suggested by Jelte Jansen in http://www.nlnetlabs.nl/svn/rfc4641bis/trunk/open-issues/Key_algorithm_roll

Added Section 4.3.5.1 Issue identified by Antoin Verschuur <http://www.nlnetlabs.nl/svn/rfc4641bis/trunk/open-issues/non-cooperative-registrars>

In Appendix A: ZSK does not nescessarily sign the DNSKEY RRset.

D.3. version 1->2

- o Significant rewrite of Section 3 whereby the argument is made that the timescakes for rollovers are made purely on operational arguments hopefully resolving http://www.nlnetlabs.nl/svn/rfc4641bis/trunk/open-issues/discussion_of_timescales

- o Added Section 5 based on <http://www.nlnetlabs.nl/svn/rfc4641bis/trunk/open-issues/NSEC-NSEC3>
- o Added a reference to draft-morris-dnsop-dnssec-key-timing [24] for the quantitative analysis on keyrolls
- o Updated Section 4.3.5 to reflect that the problem occurs when changing DNS operators, and not DNS registrars, also added the table indicating the redelegation procedure. Added text about the fact that implementations will dismiss keys that fail to validate at some point.
- o Updated a number of references.

D.4. version 2->3

- o Added bulleted list to serve as an introduction on the decision tree in Section 3.
- o In section Section 3.1:
 - * tried to motivate that keylength is not a strong motivation for KSK ZSK split (based on http://www.educatedguesswork.org/2009/10/on_the_security_of_zsk_rollove.html)
 - * Introduced Common Signing Key terminology and made the arguments for the choice of a Common Signing Key more explicit.
 - * Moved the SEP flag considerations to its own paragraph
- o In a few places in the document, but section Section 4 in particular the comments from Patrik Faltstrom (On Mar 24, 2010) on the clarity on the roles of the registrant, dns operator, registrar and registry was addressed.
- o Added some terms based on http://www.nlnetlabs.nl/svn/rfc4641bis/trunk/open-issues/timing_terminology
- o Added paragrap 2 and clarified the second but last paragraph of Section 3.2.2.
- o Clarified the table and some text in Section 4.1.5. Also added some text on what happens when the algorithm rollover also involves a roll from NSEC to NSEC3.
- o Added a paragraph about rolling KSKs that are also configured as trust-anchors in Section 4.1.2

- o Added Section 4.1.4.
- o Added Section 4.4.2 to address issue "Signature_validity"

D.5. version 3->4

- o Stephen Morris submitted a large number of language, style and editorial nits.
- o Section 4.1.5 improved based on comments from Olafur Gudmundsson and Ondrej Sury.
- o Tried to improve consistency of notation in the various rollover figures

D.6. version 4->5

- o Improved consistency of notation
- o Matthijs Mekking provided substantive feedback on algorithm rollover and suggested the content of the subsections of Section 4.1.5 and the content of the figures in Appendix C

D.7. Subversion information

www.nlnetlabs.nl/svn/rfc4641bis/

\$Id: draft-ietf-dnsop-rfc4641bis-05.txt 71 2010-10-22 19:19:43Z olaf \$

Author's Address

Olaf M. Kolkman
NLnet Labs
Kruislaan 419
Amsterdam 1098 VA
The Netherlands

EMail: olaf@nlnetlabs.nl
URI: <http://www.nlnetlabs.nl>

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 13, 2011

N. Kong
C. Wang
X. Lee
CNNIC
July 12, 2010

An Automated Synchronization Mechanism for Configuration Data of DNS
(Domain Name System) Name Servers
draft-kong-dns-conf-auto-sync-01

Abstract

This document proposes an in-band synchronization mechanism to automatically synchronize configuration data among multiple DNS (Domain Name System) name servers. Using this mechanism, any part of configuration data of a name server can be constructed as a similar DNS zone file, and be automatically synchronized by DNS messaging and notifying mechanisms.

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 <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on January 13, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

| | |
|---|----|
| 1. Introduction | 3 |
| 2. Terminology | 3 |
| 3. Definitions | 3 |
| 4. Overview | 4 |
| 5. Configuration Record | 4 |
| 5.1. Format | 5 |
| 5.2. Standard CRs | 6 |
| 5.2.1. SERIAL CR | 6 |
| 5.2.2. Other CRs | 6 |
| 6. Messages | 7 |
| 6.1. Query and Response | 7 |
| 6.1.1. Header Section | 7 |
| 6.1.2. Question Section | 9 |
| 6.1.3. Answer, Security, and Additional Section | 9 |
| 6.2. NOTIFY | 10 |
| 7. Transport | 10 |
| 8. IANA Considerations | 10 |
| 9. Security considerations | 10 |
| 10. Acknowledgements | 10 |
| 11. Change History | 11 |
| 11.1. draft-kong-dns-conf-auto-sync: Version 00 | 11 |
| 11.2. draft-kong-dns-conf-auto-sync: Version 01 | 11 |
| 12. Normative References | 11 |
| Authors' Addresses | 12 |

1. Introduction

The Domain Name System (DNS) [RFC1034] is a hierarchical naming system, which requires that more than one name server exist for every delegated domain (zone). Each name server needs to be instructed by some configuration data, which is normally used to configure the behavior and functionality of the name server, e.g. the zones that the name server needs to support, the access control list of the hosts or users that is permitted to access the name server's zone files, or the mechanism of zone transfer. Normally, these configuration data are manually managed by a local system administrator [RFC1033].

However, in order to enhance the Service Level Agreement (SLA) for DNS services, most registries, registrars and providers of the managed DNS service have implemented a fair number of name servers for their zones. Every time the policy of configuration changes, e.g. a new zone needs to be added, or DNSSEC needs to be supported, administrators need to modify a lot of configuration data of name servers within a period of time. Obviously, the manual operations of the configuration data turn into heavy burden for administrators.

This document proposes an in-band synchronization mechanism to automatically synchronize configuration data among multiple DNS name servers. Using this mechanism, any part of configuration data of a name server can be constructed as a similar DNS zone file, and be automatically synchronized by DNS messaging and notifying mechanisms. By this way, administrators don't need to manually modify every configuration files, and registries, registrars and providers of the managed DNS service don't need to develop some out-of-band softwares to realize the automatic management of configuration files for name servers.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Definitions

The following definitions are used in this document:

- o Configuration data: Some information used to control the behavior and functionality of a name server. By now, its syntax depends on the implementation of the name server.

- o Clause of Configuration: A clause which contains a serial of configuration items used to control some kind of behavior or functionality of a name server. For example, a clause is used to specify the zones that the name server needs to support, and another clause is used to confirm the access control list of the hosts or users that is permitted to access a name server's zone files. A clause of configuration is the smartest unit of synchronization.
- o Sentence of Configuration: A sentence which contains a specific configuration item used to control one behavior or functionality of a name server.
- o Configuration Record (CR): A data record used to contain the configuration data of a sentence with the similar format of DNS resource record.
- o Configuration Zone (CZ): A zone contains a series of CRs related to a clause.
- o Access Set of CZ: A set of name servers to be allowed to access (acquire) the zone data of a CZ.
- o Synchronization Set of CZ: A set of name servers to be allowed to synchronize (modify) the zone data of a CZ.
- o Notify Set of CZ: A set of name servers to be notified of changes of a CZ's zone data.

4. Overview

According to the proposed mechanism, whenever synchronization is to be done from one name server to others, several CZ files which is similar to DNS zone files are constructed and CRs are included in the files in the formats specified in this document.

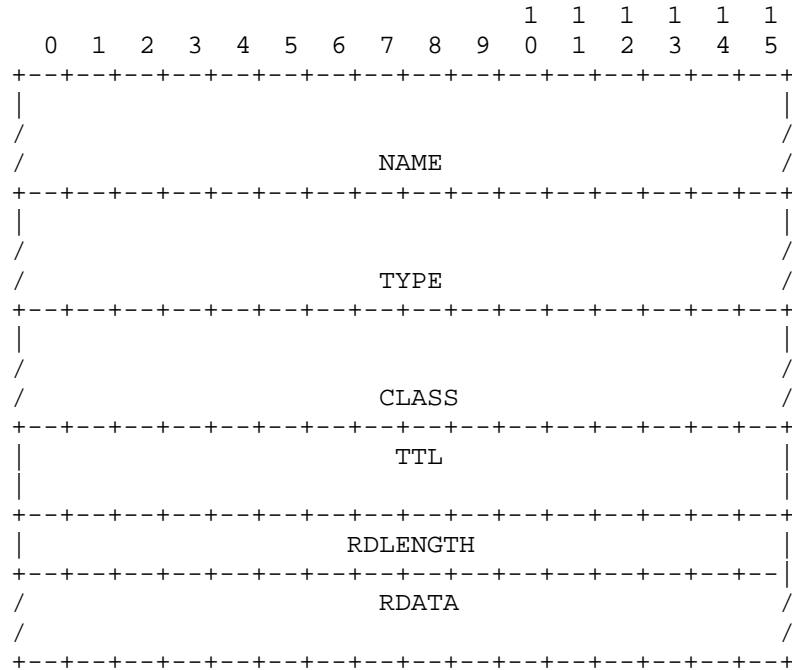
Once a CZ file has been changed, the performing name server will advise a set of other name servers within a predefined Notify Set of the change. If the identity of the notification could be verified, the notified name servers could request the new data via the current DNS messaging mechanism and do the re-configuration according to the obtained configuration data.

5. Configuration Record

The definition of CR complies with section 3.1 of [RFC1035].

5.1. Format

All CRs have the same top level format which is defined in section 3.2.1 of [RFC1035], with some fields redefined as follows:



where:

- o NAME: A name of the CZ to which this CR pertains. This fields are represented as a sequence of labels, where each label consists of a length octet followed by that number of octets.
- o TYPE: A type of the requested CR. This fields are represented as a sequence of labels, where each label consists of a length octet followed by that number of octets.
- o CLASS: A class of the CZ to which this CR pertains. This fields are represented as a sequence of labels, where each label consists of a length octet followed by that number of octets.
- o TTL: Same meaning as [RFC1035].
- o RDLENGTH: An unsigned 16 bit integer that specifies the length in octets of the CRDATA field.

- o RDATA: A variable length string of octets that describes the resource. The format of this field according to the TYPE and CLASS of the CR.

For example, there is a clause used to specify the "example.com" zone that the name server needs to support, and the mechanism of zone transfer for this zone is AXFR. Then the NAME of a CZ for this clause is "example.com", the CLASS of this kind of CZs can be defined as "Zone", the type of a CR for the mechanism of zone transfer can be defined as "zone_transfer", and the RDATA will contain the following characters "AXFR".

5.2. Standard CRs

5.2.1. SERIAL CR

Every CZ SHOULD has a standard CR named SERIAL, which used to indicate the version of a CZ.

The value of the TYPE field of a SERIAL CR must be SERIAL.

The format of the RDATA field is defined as follows:

```

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     SERIAL                             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Where:

- o SERIAL: An unsigned 32 bit version number of a CR. CR transfers preserve this value. This value wraps and should be compared using sequence space arithmetic.

5.2.2. Other CRs

The definitions of other CRs should be expected, which are used to express specific configuration items used to control some kind of behavior or functionality of a name server.

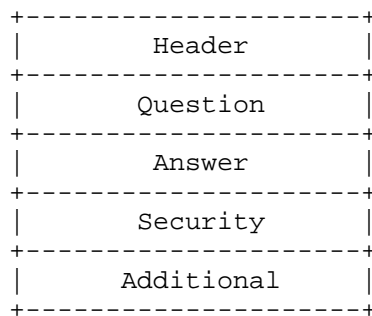
Note that the list of other CRs which are necessary to be defined here need to be discussed, e.g. the CR used to contain configuration data for DNS zones, the CR used to contain configuration data for access control list, and the CR used to contain configuration data for DNS logging, and so on. Then the new TYPE, CLASS and RDATA formats of these CRs should be defined here later.

6. Messages

This automated synchronization mechanism for the configuration data of multiple DNS name servers uses the DNS message format defined by [RFC1035], although it makes some extensions and overload some fields. Fields not otherwise described herein are to be filled with binary zero (0).

6.1. Query and Response

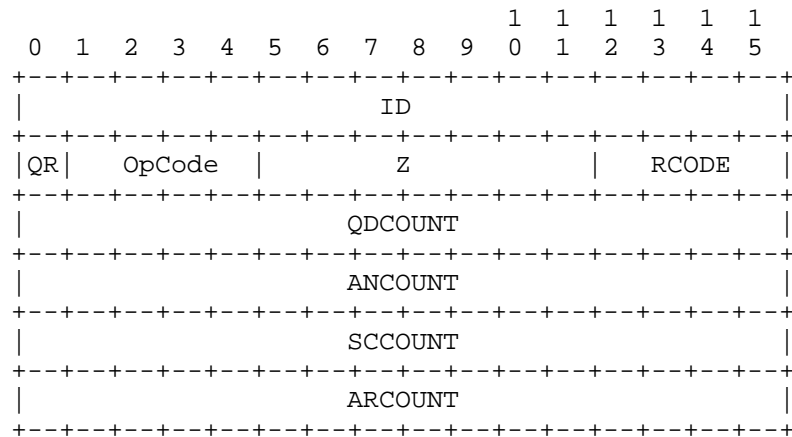
The overall format of a query message or a response message is divided into 5 sections (some of which are empty in certain cases) shown below:



The Header Section specifies that this message is a query or a response of the automated synchronization mechanism for the configuration data of multiple DNS name servers, and describes the size of the other sections. The Question Section contains fields that describe a question to a name server. The Answer Section contains CRs that answer the question. The Security Section contains some security information for authentication. The Additional Section contains some additional information which relate to the query, but are not strictly answers for the question.

6.1.1. Header Section

The header for queries or responses is based on the DNS message format which is defined in section 4.1.1 of [RFC1035], with some fields redefined as follows:



These fields are used as follows:

- o ID: Same meaning as [RFC1035].
- o QR: A one bit field that specifies whether this message is a query (0), or a response (1).
- o OpCode: A four bit field that specifies the kind of request in this message. This value is set by the originator of a request and copied into the response. The OpCode value that identifies a queries or responses message of the automated synchronization mechanism for the configuration data of multiple DNS name servers is six (6).
- o Z: Reserved for future use. Must be zero in all queries and responses.
- o RCODE: this four bit field is undefined in queries and set in responses. The values and meanings of this field within responses are as follows:
 - * 0-2: Same meaning as [RFC1035].
 - * 3: Name Error - the name of a CZ referenced in the query does not exist.
 - * 4-5: Same meaning as [RFC1035].
 - * 6-15: Reserved for future use.

- o QDCOUNT: Same meaning as [RFC1035].
- o ANCOUNT: Same meaning as [RFC1035].
- o SCCOUNT: an unsigned 16 bit integer specifying the number of configuration records used for authentication in the Security section.
- o ARCOUNT: an unsigned 16 bit integer specifying the number of configuration records in the Additional section.

6.1.2. Question Section

The Question section contains QDCOUNT (usually 1) entries, each of the following format:

```

                                1 1 1 1 1 1
                                0 1 2 3 4 5
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|
/                               QNAME                               /
/
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|
/                               QTYPE                               /
/
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|
/                               QCLASS                              /
/
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

where:

- o QNAME: the same as the NAME field of CRs.
- o QTYPE: the same as the TYPE field of CRs.
- o QCLASS: the same as the CLASS field of CRs.

6.1.3. Answer, Security, and Additional Section

The answer, authority, and additional sections all share the same format: a variable number of configuration records, where the number of records is specified in the corresponding count field in the header.

6.2. NOTIFY

By the function of NOTIFY, a name server may advise a set of other name servers within a predefined Notify Set of a CZ that the CZ's data has been changed and that a query should be initiated to request the new data.

The message format of NOTIFY is the same as the query and response. The OpCode value of NOTIFY is seven (7).

The mechanism of NOTIFY is the same as DNS NOTIFY defined by [RFC1996].

7. Transport

The transport of the messages is the same as the one of standard DNS message defined by section 4.2 of [RFC1035]. Both UDP and TCP are acceptable, but TCP is recommended.

The mechanisms for DNS zone transfer (AXFR [RFC5936] and IXFR [RFC1995]) could be used for CZ transfer.

8. IANA Considerations

IANA is requested to assign the OpCode 6 to the query and response of the automated synchronization mechanism for the configuration data of multiple DNS name servers, and the OpCode 7 to the NOTIFY of this mechanism.

9. Security considerations

Because the configuration data of a name server can be synchronized by other name servers using this automated synchronization mechanism, it's possible that the behavior and functionality of a name server will be maliciously modified by other name server. So any implementation of this document is strongly suggested to realize the Access Set of CZs and the Synchronization Set of CZs. Moreover, TSIG [RFC2845] is supposed to be used for authentication.

10. Acknowledgements

The authors especially thank the authors of [RFC1035] and [RFC1996], and the following ones: Feng Han, Guonian Sun.

11. Change History

RFC Editor: Please remove this section.

11.1. draft-kong-dns-conf-auto-sync: Version 00

- o First draft

11.2. draft-kong-dns-conf-auto-sync: Version 01

- o Improved the text.
- o Added the "Overview" section.
- o Updated the "Definitions" section. Deleted the definition of "Configuration Template".
- o Modified the example of the "Configuration Record" section. Changed "test.cn" into "example.com".
- o Added the "Other CRs" sub section in the "Configuration Record" section.
- o Added the "Change History" section.

12. Normative References

- [RFC1033] Lottor, M., "Domain administrators operations guide", RFC 1033, November 1987.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.
- [RFC1995] Ohta, M., "Incremental Zone Transfer in DNS", RFC 1995, August 1996.
- [RFC1996] Vixie, P., "A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)", RFC 1996, August 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2845] Vixie, P., Gudmundsson, O., Eastlake, D., and B. Wellington, "Secret Key Transaction Authentication for DNS

(TSIG)", RFC 2845, May 2000.

[RFC5936] Lewis, E. and A. Hoenes, "DNS Zone Transfer Protocol (AXFR)", RFC 5936, June 2010.

Authors' Addresses

Ning Kong
CNNIC
4 South 4th Street,Zhongguancun,Haidian District
Beijing, Beijing 100190
China

Phone: +86 10 5881 3147
Email: nkong@cnnic.cn

Cindy Wang
CNNIC
4 South 4th Street,Zhongguancun,Haidian District
Beijing, Beijing 100190
China

Phone: +86 10 5881 3074
Email: wangxin@cnnic.cn

Xiaodong Lee
CNNIC
4 South 4th Street,Zhongguancun,Haidian District
Beijing, Beijing 100190
China

Phone: +86 10 5881 3020
Email: lee@cnnic.cn

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 25, 2011

J. Livingood
Comcast
October 22, 2010

IPv6 AAAA DNS Whitelisting Implications
draft-livingood-dns-whitelisting-implications-01

Abstract

The objective of this document is to describe what whitelisting of DNS AAAA resource records is, or DNS whitelisting for short, as well as what the implications of this emerging practice are and what alternatives may exist. The audience for this document is the Internet community generally, including the IETF and IPv6 implementers.

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 <http://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 April 25, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as

described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

| | |
|---|----|
| 1. Introduction | 4 |
| 2. How DNS Whitelisting Works | 5 |
| 3. Concerns Regarding DNS Whitelisting | 7 |
| 4. Similarities to Split DNS | 9 |
| 5. Likely Deployment Scenarios | 10 |
| 5.1. Deploying DNS Whitelisting Universally | 10 |
| 5.2. Deploying DNS Whitelisting On An Ad Hoc Basis | 11 |
| 6. What Problems Are DNS Whitelisting Implementers Trying To Solve? | 11 |
| 7. Implications of DNS Whitelisting | 12 |
| 7.1. Architectural Implications | 12 |
| 7.2. Public IPv6 Address Reachability Implications | 13 |
| 7.3. Operational Implications | 13 |
| 7.3.1. De-Whitelisting May Occur | 13 |
| 7.3.2. Authoritative DNS Server Operational Implications | 13 |
| 7.3.3. DNS Recursive Resolver Server Operational Implications | 14 |
| 7.3.4. Monitoring Implications | 15 |
| 7.3.5. Troubleshooting Implications | 15 |
| 7.3.6. Additional Implications If Deployed On An Ad Hoc Basis | 16 |
| 7.4. Homogeneity May Be Encouraged | 16 |
| 7.5. Technology Policy Implications | 17 |
| 7.6. IPv6 Adoption Implications | 18 |
| 8. Solutions | 18 |
| 8.1. Implement DNS Whitelisting Universally | 18 |
| 8.2. Implement DNS Whitelisting On An Ad Hoc Basis | 18 |
| 8.3. Do Not Implement DNS Whitelisting | 19 |
| 8.3.1. Solving Current End User IPv6 Impairments | 19 |
| 9. Security Considerations | 19 |
| 9.1. DNSSEC Considerations | 20 |
| 9.2. Authoritative DNS Response Consistency Considerations | 20 |
| 10. IANA Considerations | 20 |
| 11. Contributors | 20 |
| 12. Acknowledgements | 21 |
| 13. References | 21 |
| 13.1. Normative References | 21 |
| 13.2. Informative References | 22 |
| Appendix A. Document Change Log | 22 |
| Appendix B. Open Issues | 23 |
| Author's Address | 23 |

1. Introduction

[EDITORIAL: This is a rough first -00 draft. Some sections have not yet been completed but will be soon. Suggestions on all parts of this document are eagerly solicited.]

This document describes the emerging practice of whitelisting of DNS AAAA resource records (RRs), or DNS whitelisting for short. It also explores the implications of this emerging practice and what alternatives may exist.

The practice of DNS whitelisting appears to have first been used by major web content sites. These web site operators observed that when they added AAAA RRs to their authoritative DNS servers that a small fraction of end users had slow or otherwise impaired access to a given web site with both AAAA and A RRs. The fraction of users with such impaired access has been estimated to be roughly 0.078% of total Internet users [IETF 77 DNSOP WG Presentation] [Network World Article on IETF 77 DNSOP WG Presentation]. Thus, in an example Internet Service Provider (ISP) network of 10 million users, approximately 7,800 of those users may experience such impaired access.

As a result of this impairment affecting end users of a given domain, a few large web site operators have begun to either implement DNS whitelisting or strongly consider the implementation of DNS whitelisting [Network World Article on DNS Whitelisting]. When implemented, DNS whitelisting in practice means that a domain's authoritative DNS will return a AAAA RR to DNS recursive resolvers [RFC1035] on the whitelist, while returning no AAAA RRs to DNS resolvers which are not on the whitelist. It is important to note that these web site operators are motivated to maintain a high-quality user experience for all of their users, and that they are attempting to shield users with impaired access from the symptoms of these impairments that would negatively affect their access to certain websites and related Internet resources.

[EDITORIAL: change web site operators --> domain operators?]

However, critics of this emerging practice of DNS whitelisting have articulated several concerns. Among these are that this is a very different behavior from the current practice concerning the publishing of IPv4 address records, that it may create a two-tiered Internet, that policies concerning whitelisting and de-whitelisting are opaque, that DNS whitelisting reduces interest in the deployment of IPv6, that new operational and management burdens are created, and that the costs and negative implications of DNS whitelisting outweigh the perceived benefits as compared to fixing underlying impairments.

This document explores the reasons and motivations for DNS whitelisting. It also explores the concerns regarding this emerging practice. As a result, readers can hopefully better understand what DNS whitelisting is, why some parties are implementing it, and why other parties are critical of the practice.

2. How DNS Whitelisting Works

DNS whitelisting is implemented in authoritative DNS servers, where those servers implement IP address-based restrictions on AAAA query responses, which contain IPv6 addresses. In practice DNS whitelisting has been primarily implemented by web server operators. For a given operator of the website `www.example.com`, that operator essentially applies an access control list (ACL) on their authoritative DNS servers, which are authoritative for the domain `example.com`. The ACL is then configured with the IPv4 and/or IPv6 addresses of DNS recursive resolvers on the Internet, which have been authorized to be added to the ACL and to therefore receive AAAA RR responses. These DNS recursive resolvers are operated by other parties, such as ISPs, universities, governments, businesses, individual end users, etc. If a DNS recursive resolver IS NOT on the ACL, then NO AAAA RRs with IPv6 addresses will be sent in response to a query for a given hostname in the `example.com` domain. However, if a DNS recursive resolver IS on the ACL, then AAAA RRs with IPv6 addresses will be sent in response to a query for a given hostname in the `example.com` domain.

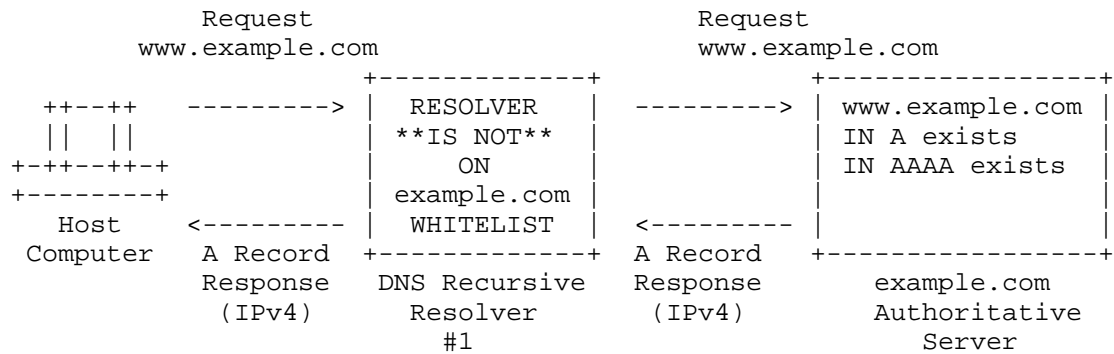
In practice this generally means that a very small fraction of the DNS recursive resolvers on the Internet can receive AAAA responses with IPv6 addresses, which means that the large majority of DNS resolvers on the Internet will receive only A RRs with IPv4 addresses. Thus, quite simply, the authoritative server hands out different answers depending upon who is asking; with IPv4 and IPv6 records for some on the authorized whitelist, and only IPv4 records for everyone else. See Figure 1 and Figure 2 for two different visual descriptions of how this works in practice.

Finally, DNS whitelisting can be deployed in two primary ways: universally on a global basis, or on an ad hoc basis. These two potential deployment models are described in Section 5.

- 1: The authoritative DNS server for example.com receives a DNS query for www.example.com, for which both A (IPv4) and AAAA (IPv6) address records exist.
- 2: The authoritative DNS server examines the IP address of the DNS recursive resolver sending the query.
- 3: The authoritative DNS server checks this IP address against the access control list (ACL) that is the DNS whitelist.
- 4: If the DNS recursive resolver's IP address IS listed in the ACL, then the response to that specific DNS recursive resolver can contain both A (IPv4) and AAAA (IPv6) address records.
- 5: If the DNS recursive resolver's IP address IS NOT listed in the ACL, then the response to that specific DNS recursive resolver can contain only A (IPv4) address records and therefore cannot contain AAAA (IPv6) address records.

Figure 1: DNS Whitelisting - System Logic

A query is sent from a DNS recursive resolver that IS NOT on the DNS whitelist:



A query is sent from a DNS recursive resolver that IS on the DNS whitelist:

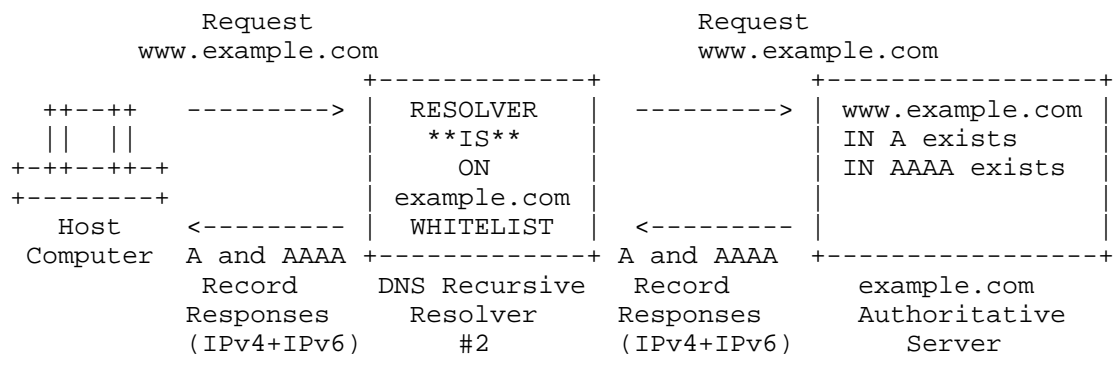


Figure 2: DNS Whitelisting - Functional Diagram

3. Concerns Regarding DNS Whitelisting

There are a number of potential implications relating to DNS whitelisting, which have raised various concerns in some parts of the Internet community. Many of those potential implications are described in Section 7.

Some parties in the Internet community are concerned that this emerging practice of DNS whitelisting for IPv6 address records could represent a departure from the generally accepted practices regarding IPv4 address records in the DNS on the Internet. These parties

explain their belief that for A records, containing IPv4 addresses, once an authoritative server operator adds the A record to the DNS, then any DNS recursive resolver on the Internet can receive that A record in response to a query. By extension, this means that any of the hosts connected to any of these DNS recursive resolvers can receive the IPv4 address records for a given FQDN. This enables new server hosts which are connected to the Internet, and for which a fully qualified domain name (FQDN) such as `www.example.com` has been added to the DNS with an IPv4 address record, to be almost immediately reachable by any host on the Internet. In this case, these new servers hosts become more and more widely accessible as new networks and new end user hosts connect to the Internet over time [EDITORIAL: consider reference to network effects]. It also means that the new server hosts do not need to know about these new networks and new end user hosts in order to make their content and applications available to them, in essence that each end in this end-to-end model is responsible for connecting to the Internet and once they have done so they can connect to each other without additional impediments or middle networks or intervening networks or servers knowing about these end points and whether one is allowed to contact the other.

In contrast, these parties are concerned that DNS whitelisting may fundamentally change this model. As a result, in this altered end-to-end model, one end (where the end user is located) cannot readily connect to the other end (where the content is located), without parts of the middle used by one end being known by the other end and approved for access to that end. Thus, as new networks connect to the Internet over time, those networks need to contact any and all domains which have implemented DNS whitelisting in order to apply to be added to their DNS whitelist, in the hopes of making the content and applications residing on named server hosts in those domains accessible by the end user hosts on that new network. Furthermore, this same need to contact all domains implementing DNS whitelisting also applies to all existing networks connected to the Internet.

Therefore, these concerned parties explain, whereas in the current IPv4 Internet when a new server host is added to the Internet it is widely available to all end user hosts and networks, when DNS whitelisting of IPv6 records is used then these new server hosts are not accessible to any end user hosts or networks until such time as the operator of the authoritative DNS servers for those new server hosts expressly authorizes access to those new server hosts by adding DNS recursive resolvers around the Internet to the ACL. This could represent a significant change in reachability of content and applications by end users and networks as these end user hosts and networks transition to IPv6. Therefore, a concern expressed is that if much of the content that end users are most interested in is not

accessible as a result, then end users and/or networks may resist adoption of IPv6 or actively seek alternatives to it, such as using multi-layer network address translation (NAT) techniques like NAT444 [I-D.shirasaki-nat444] on a long-term basis. There is also concern that this practice also could disrupt the continued increase in Internet adoption by end users if they cannot simply access new content and applications but must instead contact the operator of their DNS recursive resolver, such as their ISP or another third party, to have their DNS recursive resolver authorized for access to the content or applications that interests them. Meanwhile, these parties say, over 99.9% of all other end users that are also using that same network or DNS recursive resolver are unable to access the IPv6-based content, despite their experience being a positive one.

[EDITORIAL: Are there additional concerns to add here?]

4. Similarities to Split DNS

DNS whitelisting as described herein is in some ways similar to so-called split DNS, which is briefly described in Section 3.8 of [RFC2775]. When split DNS is used, the authoritative DNS server returns different responses depending upon what host has sent the query. While [RFC2775] notes the typical use of split DNS is to provide one answer to hosts on an Intranet and a different answer to hosts on the Internet, the essence is that different answers are provided to hosts on different networks. This is basically the way that DNS whitelisting works, in so far as hosts of different networks, which use different DNS recursive resolvers, receive different answers if one DNS recursive resolver is on the whitelist and the other is not. Thus, in a way, DNS whitelisting could in some ways be considered split DNS on the public Internet, though with some differences.

In [RFC2956], Internet transparency and Internet fragmentation concerns regarding split DNS are detailed in Section 2.1. [RFC2956] further notes in Section 2.7, concerns regarding split DNS and that it "makes the use of Fully Qualified Domain Names (FQDNs) as endpoint identifiers more complex." Section 3.5 of [RFC2956] further recommends that maintaining a stable approach to DNS operations is key during transitions such as the one to IPv6 that is underway now, stating that "Operational stability of DNS is paramount, especially during a transition of the network layer, and both IPv6 and some network address translation techniques place a heavier burden on DNS."

5. Likely Deployment Scenarios

In considering how DNS whitelisting may emerge more widely, there are two likely deployment scenarios, which are explored below.

5.1. Deploying DNS Whitelisting Universally

The least likely deployment scenario is one where DNS whitelisting becomes a standardized process across all authoritative DNS servers, across the entire Internet. While this scenario is the least likely, due to some parties not sharing the concerns that have so far motivated the use of DNS whitelisting, it is nonetheless conceivable that it could be one of the ways in which DNS whitelisting may be deployed.

In order for this deployment scenario to occur, it is likely that DNS whitelisting functionality would need to be built into all authoritative DNS server software, and that all operators of authoritative DNS servers would have to upgrade their software and enable this functionality. Furthermore, it is likely that new Internet Draft documents would need to be developed which describe how to properly configure, deploy, and maintain DNS whitelisting. As a result, it is unlikely that DNS whitelisting would, at least in the next several years, become universally deployed. Furthermore, these DNS whitelists are likely to vary on a domain-by-domain basis, depending upon a variety of factors. Such factors may include the motivation of each domain owner, the location of the DNS recursive resolvers in relation to the source content, as well as various other parameters that may be transitory in nature, or unique to a specific end user host type. Thus, it is probably unlikely that a single clearinghouse for managing whitelisting is possible; it will more likely be unique to the source content owners and/or domains which implement DNS whitelists.

While this scenario may be unlikely, it may carry some benefits. First, parties performing troubleshooting would not have to determine whether or not DNS whitelisting was being used, as it always would be in use. In addition, if universally deployed, it is possible that the criteria for being added to or removed from a DNS whitelist could be standardized across the entire Internet. Nevertheless, even if uniform DNS whitelisting policies were not standardized, it is also possible that a central registry of these policies could be developed and deployed in order to make it easier to discover them, a key part of achieving transparency regarding DNS whitelisting.

[EDITORIAL: Are there additional benefits or challenges to add here?]

5.2. Deploying DNS Whitelisting On An Ad Hoc Basis

This is the most likely deployment scenario for DNS whitelisting, as it seems today, is where some interested parties engage in DNS whitelisting but many or most others do not do so. What can make this scenario challenging from the standpoint of a DNS recursive resolver operator is determining which domains implement DNS whitelisting, particularly since a domain may not do so as they initially transition to IPv6, and may instead do so later. Thus, a DNS recursive resolver operator may initially believe that they can receive AAAA responses with IPv6 addresses as a domain adopts IPv6, but then notices via end user reports that they no longer receive AAAA responses due to that site adopting DNS whitelisting.

Thus, in contrast to universal deployment of DNS whitelisting, deployment on an ad hoc basis is likely to be significantly more challenging from an operational, monitoring, and troubleshooting standpoint. In this scenario, a DNS recursive resolver operator will have no way to systematically determine whether DNS whitelisting is or is not implemented for a domain, since the absence of AAAA records with IPv6 addresses may simply be indicative that the domain has not yet added IPv6 addressing for the domain, not that they have done so but have restricted query access via DNS whitelisting. As a result, discovering which domains implement DNS whitelisting, in order to differentiate them from those that do not, is likely to be challenging.

On the other hand, one benefit of DNS whitelisting being deployed on an ad hoc basis is that only the domains that are interested in doing so would have to upgrade their authoritative DNS servers in order to implement the ACLs necessary to perform DNS whitelisting.

[EDITORIAL: Additional benefits or challenges to add?]

6. What Problems Are DNS Whitelisting Implementers Trying To Solve?

As noted in Section 1, domains which implement DNS whitelisting are attempting to protect a few users of their domain, which happen to have impaired IPv6 access, from having a negative end user experience. While it is outside the scope of this document to explore the various reasons why a particular user may experience impaired IPv6 access, for the users which experience this it is a very real effect and would of course affect access to all or most IPv4 and IPv6 dual stack servers. This negative end user experience can range from someone slower than usual (as compared to native IPv4-based access), to extremely slow, to no access to the domain whatsoever.

Thus, parties which implement DNS whitelisting are attempting to provide a good experience to these end users. While one can debate whether DNS whitelisting is the optimal solution, it is quite clear that DNS whitelisting implementers are extremely interested in the performance of their services for end users as a primary motivation.

[EDITORIAL 1: More motivations to add?]

[EDITORIAL 2: Any good external references to consider adding?]

7. Implications of DNS Whitelisting

There are many potential implications of DNS whitelisting. In the sections below, the key potential implications are listed in some detail.

7.1. Architectural Implications

DNS whitelisting could be perceived as somewhat modifying the end-to-end model that prevails on the IPv4 Internet today. This approach moves additional access control information and policies into the middle of the network on the IPv6-addressed Internet, which did not exist before on the IPv4-addressed Internet. This could raise some risks noted in [RFC3724], which in explaining the history of the end-to-end principle [RFC1958] explains that one of the goals is to minimize the state, policies, and other functions needed in the middle of the network in order to enable end-to-end communications on the Internet.

It is also possible that DNS whitelisting could place at risk some of the benefits of the end-to-end principle, as listed in Section 4.1 of [RFC3724], such as protection of innovation. Further, while [RFC3234] details issues and concerns regarding so-called middleboxes, there may be parallels to DNS whitelisting, especially concerning modified DNS servers noted in Section 2.16 of [RFC3234], and more general concerns noted in Section 1.2 of [RFC3234] about the introduction of new failure modes, that configuration is no longer limited to two ends of a session, and that diagnosis of failures and misconfigurations is more complex.

In [Tussle in Cyberspace], the authors note concerns regarding the introduction of new control points, as well as "kludges" to the DNS, as risks to the goal of network transparency in the end-to-end model. Some parties concerned with the emerging use of DNS whitelisting have shared similar concerns, which may make [Tussle in Cyberspace] an interesting and relevant document. In addition, [Rethinking the design of the Internet] reviews similar issues that may be of

interest to readers of this document.

In order to explore and better understand these high-level architectural implications and concerns in more detail, the following sections explore more specific potential implications.

7.2. Public IPv6 Address Reachability Implications

The predominant experience of end user hosts and servers on the IPv4-addressed Internet today is that, very generally speaking, when a new server with a public IPv4 address is added, that it is then globally accessible by IPv4-addressed hosts. For the purposes of this document, that concept can be considered "pervasive reachability". It has so far been assumed that the same expectations of reachability would exist in the IPv6-addressed Internet. However, if DNS whitelisting is deployed, this will not be the case since only end user hosts using DNS recursive resolvers which have been added to the ACL of a given domain using DNS whitelisting would be able to reach new servers in that given domain via IPv6 addresses.

Thus, the expectation of any end user host being able to connect to any server (essentially both hosts, just at either end of the network), defined here as "pervasive reachability", will change to "restricted reachability" with IPv6.

[EDITORIAL: Additional implications?]

7.3. Operational Implications

This section explores some of the operationally related implications which may occur as a result of, related to, or necessary when engaging in the practice of DNS whitelisting.

7.3.1. De-Whitelisting May Occur

If it is possible for a DNS recursive resolver to be added to a whitelist, then it is also possible for that resolver to be removed from the whitelist, also known as de-whitelisting. Since de-whitelisting can occur, whether through a decision by the authoritative server operator or the domain owner, or even due to a technical error, an operator of a DNS recursive resolver will have new operational and monitoring requirements and/or needs as noted in Section 7.3.3, Section 7.3.4, Section 7.3.5, and Section 7.5.

7.3.2. Authoritative DNS Server Operational Implications

Operators of authoritative servers may need to maintain an ACL a server-wide basis affecting all domains, on a domain-by-domain basis,

as well as on a combination of the two. As a result, operational practices and software capabilities may need to be developed in order to support such functionality. In addition, processes may need to be put in place to protect against inadvertently adding or removing IP addresses, as well as systems and/or processes to respond to such incidents if and when they occur. For example, a system may be needed to record DNS whitelisting requests, report on their status along a workflow, add IP addresses when whitelisting has been approved, remove IP addresses when they have been de-whitelisted, log the personnel involved and timing of changes, schedule changes to occur in the future, and to roll back any inadvertent changes.

Such operators may also need implement new forms of monitoring in order to apply change control, as noted briefly in Section 7.3.4.

[EDITORIAL: Additional implications?]

7.3.3. DNS Recursive Resolver Server Operational Implications

Operators of DNS recursive resolvers, which may include ISPs, enterprises, universities, governments, individual end users, and many other parties, are likely to need to implement new forms of monitoring, as noted briefly in Section 7.3.4. But more critically, such operators may need to add people, processes, and systems in order to manage countless DNS whitelisting applications, for all domains that the end users of such servers are interested in now or in which they may be interested in the future. As such anticipation of interesting domains is likely infeasible, it is more likely that such operators may either choose to only apply to be whitelisted for a domain based upon one or more end user requests, or that they will attempt to do so for all domains.

When such operators apply for DNS whitelisting for all domains, that may mean doing so for all registered domains. Thus, some system would have to be developed to discover whether each domain has been whitelisted or not, which is touched on in Section 5 and may vary depending upon whether DNS whitelisting is universally deployed or is deployed on an ad hoc basis.

Furthermore, these operators will need to develop processes and systems to track the status of all DNS whitelisting applications, respond to requests for additional information related to these applications, determine when and if applications have been denied, manage appeals, and track any de-whitelisting actions. Given the incredible number of domains in existence, the ease with which a new domain can be added, and the continued strong growth in the numbers of new domains, readers should not underestimate the potential significance in personnel and expense that this could represent for

such operators. In addition, it is likely that systems and personnel may also be needed to handle new end user requests for domains for which to apply for DNS whitelisting, and/or inquiries into the status of a whitelisting application, reports of de-whitelisting incidents, general inquiries related to DNS whitelisting, and requests for DNS whitelisting-related troubleshooting by these end users.

[EDITORIAL: Additional implications?]

7.3.4. Monitoring Implications

Once a DNS recursive resolver has been whitelisted for a particular domain, then the operator of that DNS recursive resolver may need to implement monitoring in order to detect the possible loss of whitelisting status in the future. This DNS recursive resolver operator could configure a monitor to check for a AAAA response in the whitelisted domain, as a check to validate continued status on the DNS whitelist. The monitor could then trigger an alert if at some point the AAAA responses were no longer received, so that operations personnel could begin troubleshooting, as outlined in Section 7.3.5.

Also, authoritative DNS server operators are likely to need to implement new forms of monitoring. In this case, they may desire to monitor for significant changes in the size of the whitelist within a certain period of time, which might be indicative of a technical error such as the entire ACL being removed. These operators may also wish to monitor their workflow process for reviewing and acting upon DNS whitelisting applications and appeals, potentially measuring and reporting on service level commitments regarding the time an application or appeal can remain at each step of the process, regardless of whether or not such information is shared with parties other than that authoritative DNS server operator.

These are but a few examples of the types of monitoring that may be called for as a result of DNS whitelisting, among what are likely many other types and variations.

[EDITORIAL: Additional implications?]

7.3.5. Troubleshooting Implications

The implications of DNS whitelisted present many challenges, which have been detailed in Section 7. These challenges may negatively affect the end users' ability to troubleshoot, as well as that of DNS recursive resolver operators, ISPs, content providers, domain owners (where they may be different from the operator of the authoritative DNS server for their domain), and other third parties. This may make

the process of determining why a server is not reachable significantly more complex.

[SECTION INCOMPLETE - MIGHT LIKE TO ADD SOME EXAMPLES HERE]

[EDITORIAL: Additional implications?]

7.3.6. Additional Implications If Deployed On An Ad Hoc Basis

[SECTION INCOMPLETE - IS THIS NEEDED? - PLACEHOLDER FOR NOW]

[EDITORIAL: Additional implications?]

7.4. Homogeneity May Be Encouraged

A broad trend which has existed on the Internet appears to be a move towards increasing levels of heterogeneity. One manifestation of this is in an increasing number, variety, and customization of end user hosts, including home network, operating systems, client software, home network devices, and personal computing devices. This trend appears to have had a positive effect on the development and growth of the Internet. A key facet of this that has evolved is the ability of the end user to connect any technically compliant device or use any technically compatible software to connect to the Internet. Not only does this trend towards greater heterogeneity reduce the control which is exerted in the middle of the network, described in positive terms in [Tussle in Cyberspace], [Rethinking the design of the Internet], and [RFC3724], but it can also help to enable greater and more rapid innovation at the edges.

An unfortunate implication of the adoption of DNS whitelisting may be the encouragement of a reversal of this trend, which would be a move back towards greater levels of homogeneity. In this case, a domain owner which has implemented DNS whitelisting may prefer greater levels of control be exerted over end user hosts (which broadly includes all types of end user software and hardware) in order to attempt to enforce technical standards relating to establishing certain IPv6 capabilities or the enforcing the elimination of or restriction of certain end user hosts. While the domain operator is attempting to protect, maintain, and/or optimize the end user experience for their domain, the collective result of many domains implementing DNS whitelisting, or even a few important domains implementing DNS whitelisting, may be to encourage a return to more homogenous and/or controlled end user hosts. Unfortunately, this could have unintended side effects on and counter-productive implications for future innovation at the edges of the network.

7.5. Technology Policy Implications

A key technology policy implication concerns the policies relating to the process of reviewing an application for DNS whitelisting, and the decision-making process regarding whitelisting for a domain. Important questions may include whether these policies have been fully and transparently disclosed, are non-discriminatory, and are not anti-competitive. A related implication is whether and what the process for appeals is, when a domain decides not to add a DNS recursive resolver to the whitelist. Key questions here may include whether appeals are allowed, what the process is, what the expected turn around time is, and whether the appeal will be handled by an independent third party or other entity/group.

A further implications arises when de-whitelisting occurs. Questions that may naturally be raised in such a case include whether the criteria for de-whitelisting have been fully and transparently disclosed, are non-discriminatory, and are not anti-competitive. Additionally, the question of whether or not there was a cure period available prior to de-whitelisting, during which troubleshooting activities, complaint response work, and corrective actions may be attempted, and whether this cure period was a reasonable amount of time.

It is also conceivable that whitelisting and de-whitelisting decisions could be quite sensitive to concerned parties beyond the operator of the domain which has implemented DNS whitelisting and the operator of the DNS recursive resolver, including end users, application developers, content providers, advertisers, public policy groups, governments, and other entities, which may also seek to become involved in or express opinions concerning whitelisting and/or de-whitelisting decisions. Lastly, it is conceivable that any of these interested parties or other related stakeholders may seek redress outside of the process a domain has establishing for DNS whitelisting and de-whitelisting.

A final concern is that decisions relating to whitelisting and de-whitelisting may occur as an expression of other commercial, governmental, and/or cultural conflicts, given the new control point which has be established with DNS whitelisting. For example, in one imagined scenario, it may be conceivable that one government is unhappy with a news story or book published in a particular country, and that this government may retaliate against or protest this news story or book by requiring domains operating within that government's territory to de-whitelist commercial, governmental, or other entities involved in or related to (however tangentially) publishing the news story or book. By the same token, a news site operating in multiple territories may be unhappy with governmental policies in one

particular territory and may choose to express dissatisfaction in that territory by de-whitelisting commercial, governmental, or other entities in that territory. Thus, it seems possible that DNS whitelisting and de-whitelisting could become a vehicle for adjudicating other disputes, and that this may well have intended and unintended consequences for end users which are affected by such decisions and are unlikely to be able to express a strong voice in such decisions.

7.6. IPv6 Adoption Implications

As noted in Section 3, the implications of DNS whitelisting may drive end users and/or networks to delay, postpone, or cancel adoption of IPv6, or to actively seek alternatives to it. Such alternatives may include the use of multi-layer network address translation (NAT) techniques like NAT444 [I-D.shirasaki-nat444], which these parties may decide to pursue on a long-term basis to avoid the perceived costs and aggravations related to DNS whitelisting. This could of course come at the very time that the Internet community is trying to get these very same parties interested in IPv6 and motivated to begin the transition to IPv6. As a result, parties concerned over the negative implications of DNS whitelisting have said they are very concerned of the negative effects that this practice could have on the adoption of IPv6 if it became widespread or was adopted by key parties in the Internet ecosystem.

[EDITORIAL: Additional implications?]

8. Solutions

8.1. Implement DNS Whitelisting Universally

One obvious solution is to implement DNS whitelisted universally, and to do so using some sort of centralized registry of DNS whitelisting policies, contracts, processes, or other information. This potential solution seems unlikely at the current time.

[EDITORIAL: More to add?]

8.2. Implement DNS Whitelisting On An Ad Hoc Basis

If DNS whitelisting was to be adopted more widely, it is likely to be adopted on this ad hoc, or domain-by-domain basis. Therefore, only those domains interested in DNS whitelisting would need to adopt the practice, though as noted herein discovering that they a given domain has done so may be problematic.

[EDITORIAL: More to add?]

8.3. Do Not Implement DNS Whitelisting

As an alternative to adopting DNS whitelisting, the Internet community can instead choose to take no action whatsoever, perpetuating the current predominant authoritative DNS operational model on the Internet, and leave it up to end users with IPv6-related impairments to discover and fix those impairments.

8.3.1. Solving Current End User IPv6 Impairments

A further extension of not implementing DNS whitelisting, is to also endeavor to actually fix the underlying technical problems that have prompted the consideration of DNS whitelisting in the first place, as an alternative to trying to apply temporary workarounds to avoid the symptoms of underlying end user IPv6 impairments. A first step is obviously to identify which users have such impairments, which would appear to be possible, and then to communicate this information to end users. Such end user communication is likely to be most helpful if the end user is not only alerted to a potential problem but is given careful and detailed advice on how to resolve this on their own, or where they can seek help in doing so.

One challenge with this option is the potential difficulty of motivating members of the Internet community to work collectively towards this goal, sharing the labor, time, and costs related to such an effort. Of course, since just such a community effort is now underway for IPv6, it is possible that this would call for only a moderate amount of additional work.

[EDITORIAL: More to add?]

9. Security Considerations

There are no particular security considerations if DNS whitelisting is not adopted, as this is how the public Internet works today with A records.

However, if DNS whitelisting is adopted, organizations which apply DNS whitelisting policies in their authoritative servers should have procedures and systems which do not allow unauthorized parties to either remove whitelisted DNS resolvers from the whitelist or add non-whitelisted DNS resolvers to the whitelist. Should such unauthorized additions or removals from the whitelist can be quite damaging, and result in content providers and/or ISPs to incur substantial support costs resulting from end user and/or customer

contacts. As such, great care must be taken to control access to the whitelist for an authoritative server.

In addition, two other key security-related issues should be taken into consideration:

9.1. DNSSEC Considerations

DNS security extensions defined in [RFC4033], [RFC4034], and [RFC4035] use cryptographic digital signatures to provide origin authentication and integrity assurance for DNS data. This is done by creating signatures for DNS data on a Security-Aware Authoritative Name Server that can be used by Security-Aware Resolvers to verify the answers. Since DNS whitelisting is implemented on an authoritative server, which provides different answers depending upon which resolver server has sent a query, the DNSSEC chain of trust is not altered. Therefore there are no DNSSEC implications per se, and thus no specific DNSSEC considerations to be listed.

9.2. Authoritative DNS Response Consistency Considerations

[INCOMPLETE!!]

While Section 9.1 does not contain any specific DNSSEC considerations. However, it is certainly conceivable that security concerns may arise when end users or other parties notice that the responses sent from an authoritative DNS server appear to vary from one network or one DNS recursive resolver to another. This may give rise to concerns that, since the authoritative responses vary that there is some sort of security issue and/or some or none of the responses can be trusted.

10. IANA Considerations

There are no IANA considerations in this document.

11. Contributors

The following people made significant textual contributions to this document and/or played an important role in the development and evolution of this document:

John Brzozowski

Chris Griffiths

Tom Klieber

Yiu Lee

Rich Woundy

12. Acknowledgements

The authors and contributors also wish to acknowledge the assistance of the following individuals in helping us to develop and/or review this document:

13. References

13.1. Normative References

- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.
- [RFC1958] Carpenter, B., "Architectural Principles of the Internet", RFC 1958, June 1996.
- [RFC2775] Carpenter, B., "Internet Transparency", RFC 2775, February 2000.
- [RFC2956] Kaat, M., "Overview of 1999 IAB Network Layer Workshop", RFC 2956, October 2000.
- [RFC3234] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, February 2002.
- [RFC3724] Kempf, J., Austein, R., and IAB, "The Rise of the Middle and the Future of End-to-End: Reflections on the Evolution of the Internet Architecture", RFC 3724, March 2004.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.
- [RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.
- [RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.

13.2. Informative References

- [I-D.shirasaki-nat444]
Yamagata, I., Shirasaki, Y., Nakagawa, A., Yamaguchi, J.,
and H. Ashida, "NAT444", draft-shirasaki-nat444-02 (work
in progress), July 2010.
- [IETF 77 DNSOP WG Presentation]
Gashinsky, I., "IPv6 & recursive resolvers: How do we make
the transition less painful?", IETF 77 DNS Operations
Working Group, March 2010,
<<http://www.ietf.org/proceedings/77/slides/dnsop-7.pdf>>.
- [Network World Article on DNS Whitelisting]
Marsan, C., "Google, Microsoft, Netflix in talks to create
shared list of IPv6 users", Network World , March 2010, <<http://www.networkworld.com/news/2010/032610-dns-ipv6-whitelist.html>>.
- [Network World Article on IETF 77 DNSOP WG Presentation]
Marsan, C., "Yahoo proposes 'really ugly hack' to DNS",
Network World , March 2010, <<http://www.networkworld.com/news/2010/032610-yahoo-dns.html>>.
- [Rethinking the design of the Internet]
Blumenthal, M. and D. Clark, "Rethinking the design of the
Internet: The end to end arguments vs. the brave new
world", ACM Transactions on Internet Technology Volume 1,
Number 1, Pages 70-109, August 2001, <http://dspace.mit.edu/bitstream/handle/1721.1/1519/TPRC_Clark_Blumenthal.pdf>.
- [Tussle in Cyberspace]
Braden, R., Clark, D., Sollins, K., and J. Wroclawski,
"Tussle in Cyberspace: Defining Tomorrow's Internet",
Proceedings of ACM Sigcomm 2002, August 2002, <<http://groups.csail.mit.edu/ana/Publications/PubPDFs/Tussle2002.pdf>>.

Appendix A. Document Change Log

- [RFC Editor: This section is to be removed before publication]
- 00: First version published
- 01: Updated the title of the document, to avoid confusion (based on
feedback)

Appendix B. Open Issues

[RFC Editor: This section is to be removed before publication]

1. Incorporate any feedback received at IETF 79
2. Incorporate feedback from Erik Kline, received 10/1/2010
3. Incorporate feedback from Brian Carpenter, received 10/19/2010
4. Bring on new contributors: Hannes Tschofenig and Danny McPherson has so far offered to contribute.
5. Close out any EDITORIAL notes
6. Add any good references throughout the document
7. Add reviewers to the acknowledgements section
8. Ensure references are in the proper section (normative/informative)
9. Include a number of references from RFC3724?
10. Call DNS WL something else or add note to the effect that this is unrelated to DNS WL used for email - such as www.dnswl.org

Author's Address

Jason Livingood
Comcast Cable Communications
One Comcast Center
1701 John F. Kennedy Boulevard
Philadelphia, PA 19103
US

Email: jason_livingood@cable.comcast.com
URI: <http://www.comcast.com>

