DNSOP
Internet-Draft
Obsoletes: 2541 (if approved)
Intended status: Informational
Expires: September 12, 2011

O. Kolkman
W. Mekking
NLnet Labs

March 11, 2011

DNSSEC Operational Practices, Version 2
draft-ietf-dnsop-rfc4641bis-06

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.

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.

Table of Contents

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.

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. Section 4.1 and Section 4.2 deal with the rollover, or replacement, of keys. Section 4.3 discusses considerations on how parents deal with their children's public keys in order to maintain chains of trust. Section 4.4 covers all kinds of timing issues around keys publication. Section 5 covers the considerations regarding selecting and using NSEC and NSEC3.

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 obsoletes RFC 4641 [14].

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:

- "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.

- "Signature publication period" The period that a signature is published. It starts at the time the signature is introduced in the zone for the first time and ends at the time when the signature is removed or replaced with a new signature. 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.

- "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.

- "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 then a number of decisions need to be made:

- Does one differentiate between Zone Signing and Key Signing Keys or is the use of one type of key sufficient?
- Are Key Signing Keys (likely to be) in use as Trust Anchors?
- What are the timing parameters that are allowed by the operational requirements?
- 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 to have the role of Key Signing Keys (KSKs). These keys will only sign the apex DNSKEY RRSet in a zone. Other keys can be used to sign all the other 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, keys have both the role of KSK and ZSK, 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 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.

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.

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. However, storing keys off-line or with more limitation on access control has a negative effect on the operational flexibility. 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. 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 a better protection against key compromise, without losing much operational flexibility. It must be said that some HSMs give the option to have your keys online, giving more protection and hardly affecting the the operational flexibility. In those cases, a KSK-ZSK split is not more beneficial than the Single-Type signing scheme.

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:

- the exposure to risk is low (e.g. when keys are stored on HSMs);
- one can be certain that a key is not used as a trust-anchor;
- maintenance of the various keys cannot be performed through tools (is prone to human error); and
- 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

A key that acts only as a Zone Signing Key can be used to sign all the data but the DNSKEY RRset in a zone on a regular basis. When a ZSK is to be rolled, no interaction with the parent is needed. This allows for signature validity periods on the order of days.

A key with only the Key Signing Key role is to be used to sign the DNSKEY RRs in a zone. If a KSK is to be rolled, there may be interactions with other parties. 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:

- It should be done frequently and regularly (possibly every few months) so that a key rollover remains an operational routine.
- 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.
- 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 keys that are used as KSKs and not on keys that are used as 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 one year, meaning you have the intent to replace them after 12 months. The key effectivity period is merely a policy parameter, and should not be considered a constant value. For example, the real key effectivity period may be a little bit longer than 12 months, because not all actions needed to complete the rollover could be finished in time.

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 very short key effectivity periods are theoretically possible, when replacing keys one has to take into account the rollover considerations from Section 4.1 and Section 4.4. Key replacement endures for a couple of Zone TTLs, depending on the rollover scenario. Therefore, a multiple of Zone TTL is a reasonable lower limit on the key effectivity period. Forcing a smaller key effectivity period will result your zone to have a ever-growing keyset.

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 weakened.

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

At the time of writing, there are three types of signature algorithms that can be used in DNSSEC: RSA, DSA and GOST. Proposals for other algorithms are in the making. All three are fully specified in many freely-available documents, and 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. Also, DSA in context of DNSSEC is limited to the maximum of 1024 bit keys.

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

This section assumes RSA keys, as suggested in the previous section.

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).

Depending on local policy (e.g. owners of keys that are used as extremely high value trust anchors, or non-anchor keys that may be difficult to roll over), you 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]). Signing and verifiying with a 2048-bit key takes of course longer than with a 1024-bit key. The increase depends on software and hardware implementations, but public operations (such as verification) are about four times slower, while private operations (such as signing) slow down about eight times.

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 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], or any other update mechanism that runs at a regular interval 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 updates is unavailable from general hosts on the Internet; it is not listed in the NS RRSet, although its name appears in the SOA RRs MNAME field. The nameservers in the NS RRSet 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 an HSM will be influenced by a tradeoff between various concerns:

- The risks that an unauthorized person has unnoticed read-access to the private key
- The remaining window of opportunity for the attacker.
- The economic impact of the possible attacks (for a TLD that impact will typically be higher than for an individual users).
- 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.)
- 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 800-90 [19]. In particular, one should carefully assess whether the random number generator used during key generation adheres to these suggestions. Typically, HSMs tend to provide a good facility for key generation.

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 than 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:

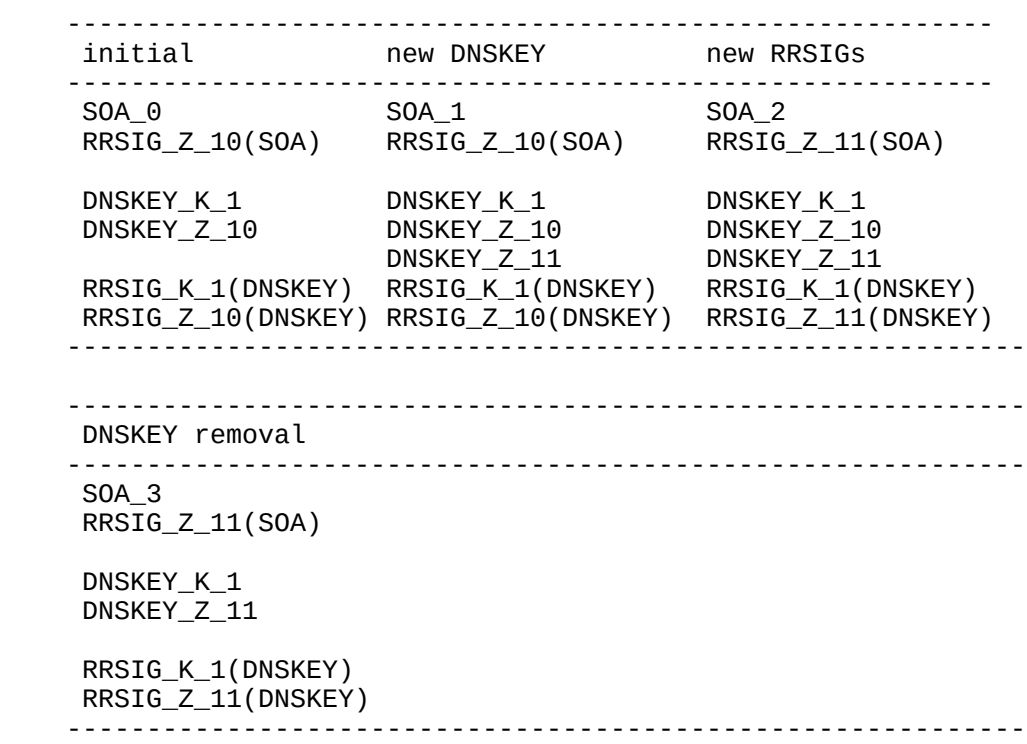


Figure 1: Pre-Publish Key Rollover

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
SOA_0	SOA_1	SOA_2
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)
SOA_3	SOA_4
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
SOA_0	SOA_1	SOA_2
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.

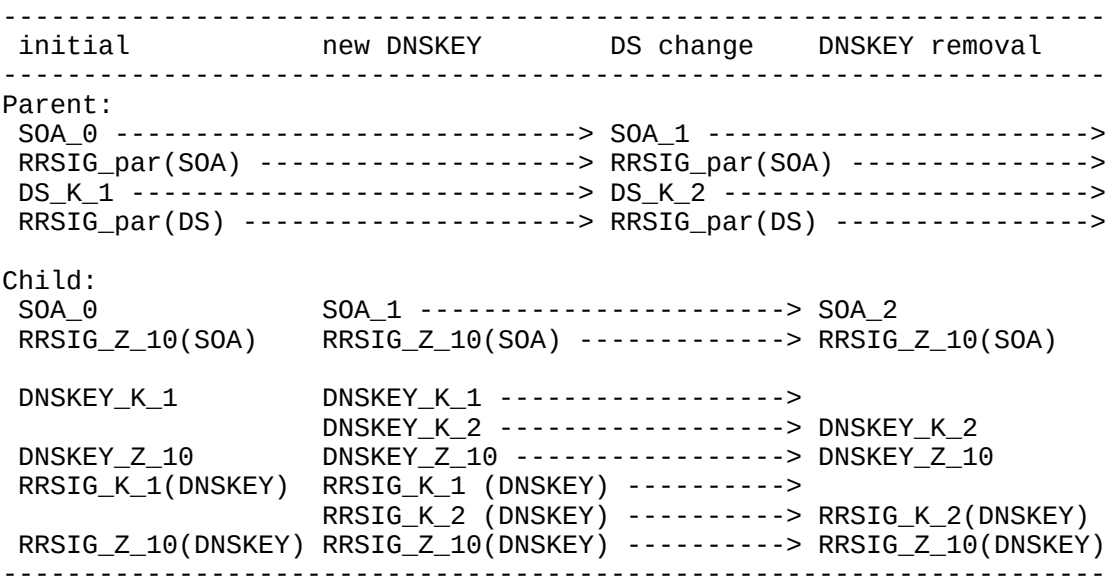


Figure 4: Stages of Deployment for a Double Signature Key Signing Key Rollover

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, meaningful: 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. A Pre-Publish method is also possible for KSKs, known as the Double-DS rollover. The name being a give away, the record that needs to be pre-published is the DS RR at the parent. The Pre-Publish method has some drawbacks for KSKs. We first describe the rollover scheme and then indicate these drawbacks.

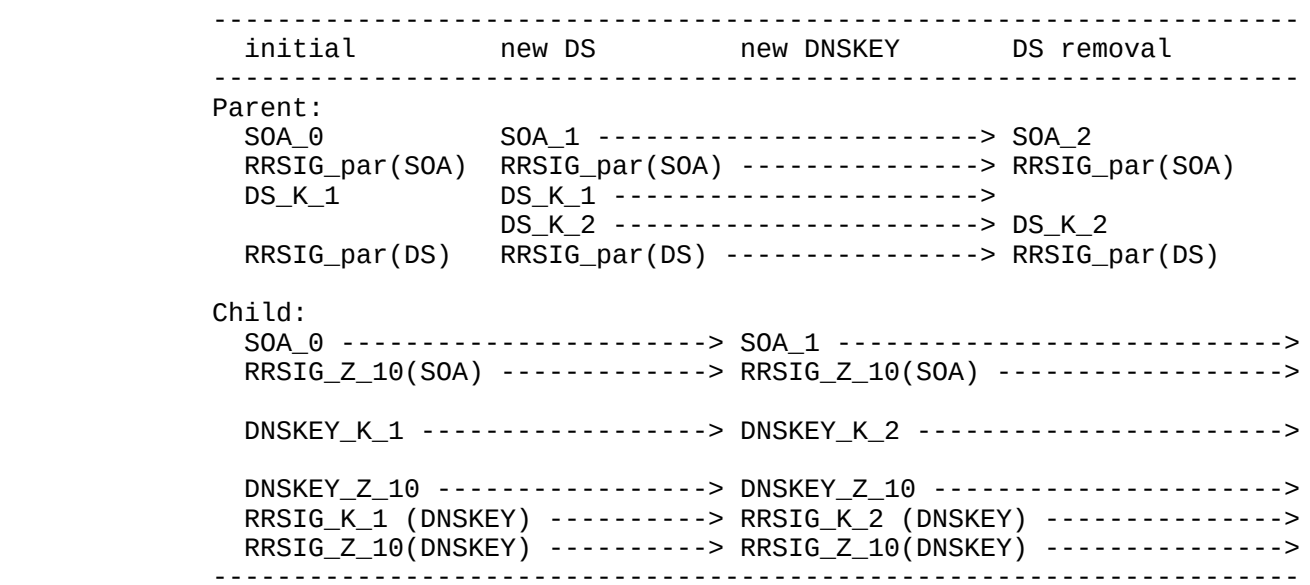


Figure 5: Stages of Deployment for a Dobule-DS Key Signing Key Rollover

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 DNSKEY_K_2. 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.

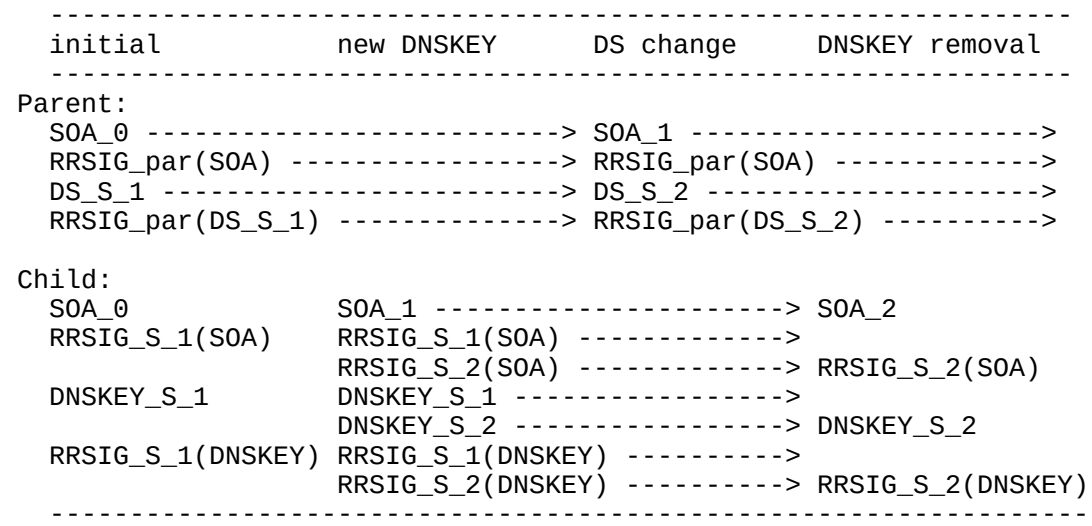


Figure 6: Stages of the Straightforward rollover in a Single Type Signing scheme

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 keyset 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 and single type keys are discussed below.

There exist a conservative and a liberal approach for algorithm rollover. This has to do with section 2.2 in RFC4035 [5]:

> There MUST be an RRSIG for each RRset using at least one DNSKEY of each algorithm in the zone apex DNSKEY RRset. The apex DNSKEY RRset itself MUST be signed by each algorithm appearing in the DS RRset located at the delegating parent (if any).

The conservative approach interprets this section very strict, meaning that it expects that all RRset has a valid signature for every algorithm signalled by the zone apex DNSKEY RRset, no matter where this RRset is kept. Important to know is that this also includes the resolvers cache. The liberal approach uses a more loose interpretation of the section and limits the rule to RRsets in the zone at the authoritative name servers. There is a reasonable argument for saying that this is valid, because the specific section is a subsection of section 2. in RFC4035: Zone Signing.

When following the more liberal approach, algorithm rollover is just as easy as a regular Double-Signature KSK rollover (Section 4.1.2). Note that the Double-DS rollover method cannot be used, since that would introduce a parental DS of which the apex DNSKEY RRset has not been signed with the introduced algorithm.

However, there are implementations of validators known that follow the more conservative approach. Performing a Double-Signature KSK algorithm rollover will temporarily make your zone appear as Bogus by such validators during the rollover. Therefore, the rollover in this section will explain the stages of deployment assuming the conservative approach.

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.

Figure 7 describes the steps. The 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(SOA) is the RRSIG RR made with the private part of the new zone signing key of algorithm type 2 over a SOA RR. It is assumed that the key that signs the SOA RR also signes all other non-DNSKEY RRset data.

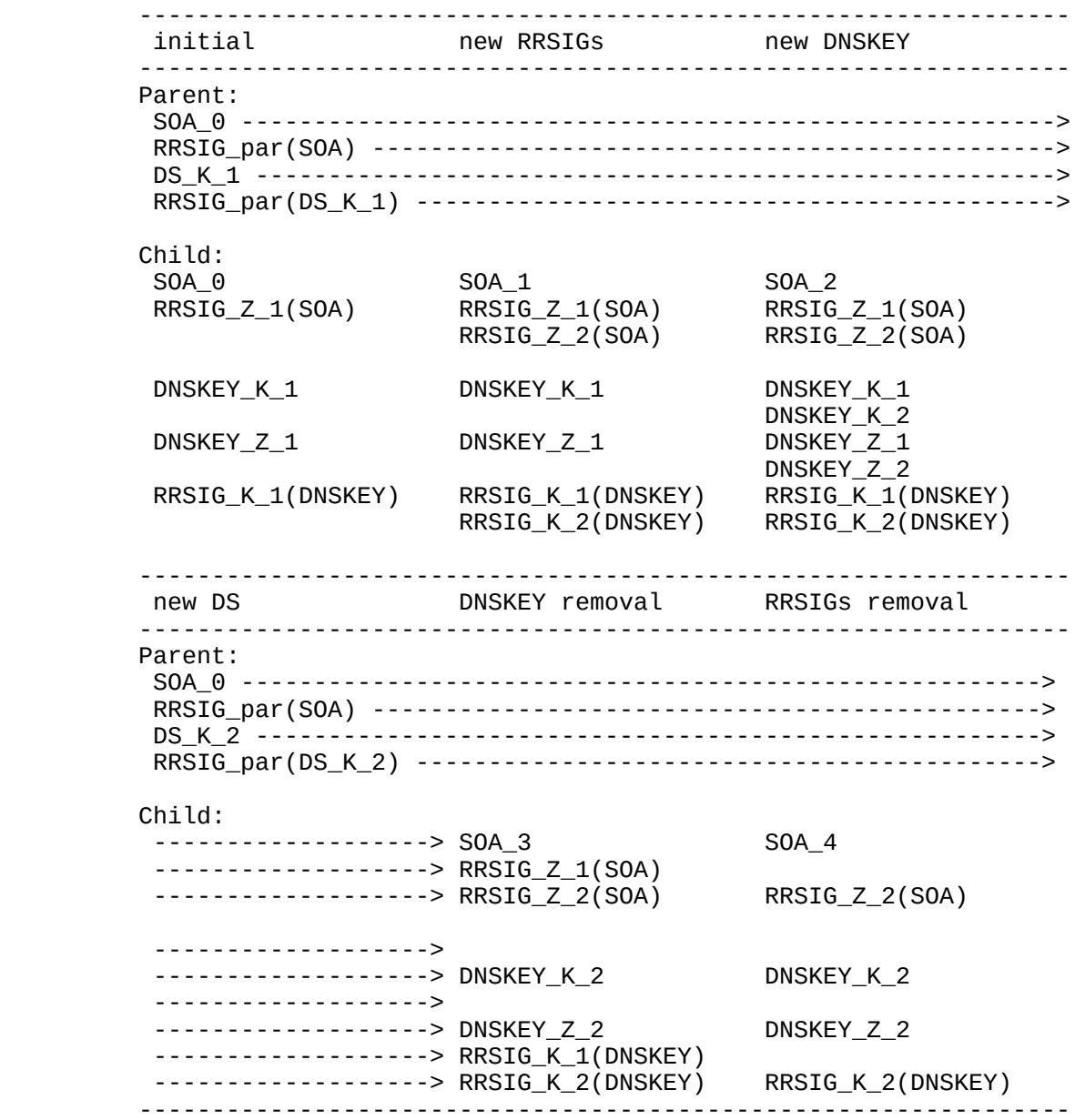


Figure 7: Stages of Deployment during an Algorithm Rollover

initial: 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.

new RRSIGs: 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.

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

new DS: 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.

DNSKEY removal: 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.

RRSIGs removal: 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: Single Type Signing Scheme Algorithm Rollover : when you have chosen not to differentiate between Zone and Key signing keys (Section 4.1.5.1)

2: RFC5011 Algorithm Rollover : when trust-anchors can track the roll via RFC5011 style rollover (Section 4.1.5.2)

3: 1 and 2 combined : when a Single Type Signing Scheme Algorithm rollover is RFC5011-enabled (Section 4.1.5.3)

In addition to the narrative below these special cases are represented in Figure 11, Figure 12 and Figure 13 in Appendix C.

4.1.5.1. 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 have the opportunity to expire from distant caches before introducing the new algorithm in the DNSKEY RRset is still valid.

Also see Figure 11 in Appendix C.

4.1.5.2. Algorithm rollover, RFC5011 style

Trust anchor algorithm rollover is almost 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 the "new DS" step, we need an additional step where the DNSKEY is revoked (revoke DNSKEY):

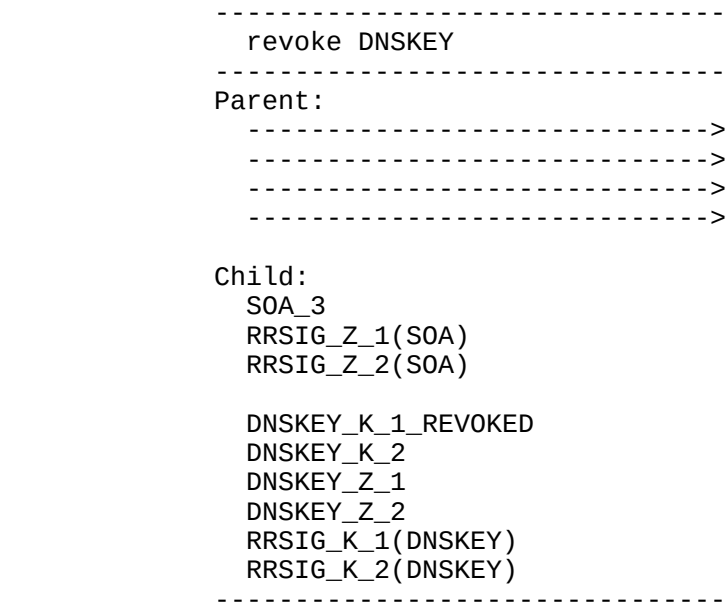


Figure 8: The Revoke DNSKEY state that is added to an algorithm rollover when RFC5011 is in use.

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 if it was DNSKEY_K_1. In other words, they will handle the revoked key as a normal key, and thus RRsets signed with this key will validate. As a result, the signature matches the algorithm listed in the DNSKEY RRset. Resolvers that do 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.

Also see Figure 12 in Appendix C.

4.1.5.3. Single Signing Type Algorithm Rollover, RFC5011 style

Combining the Single Signing Type Scheme Algorithm Rollover 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 2.1, 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 KSK/ZSK split 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. Thus, more operations are 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 than temporarily act as ZSK during the algorithm rollover.

Just like with algorithm rollover RFC5011 style, while all zone data must be signed with an unrevoked key, it is permissible to sign the keyset with a revoked key, for the same esoteric argument described in Section 4.1.5.2.

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.

4.1.5.4. 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.

4.1.6. Considerations for Automated Key Rollovers

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

- ZSK rollovers are easy to automate as only the child zone is involved.

- 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

- as long as a signature over the compromised key in the trust chain is valid,
- as long as the DS RR in the parent zone points to the compromised key,
- 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 RRSet 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. Lower the TTL for DNSKEYs so that it will expire faster from caches.

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. This provides an upper limit on how long the compromised KSK can be used in a replay attack.

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" TTL and signature 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. Note that until that time, the child zone is still vulnerable to spoofing: the attacker is still in possesion of the compromised key that the DS points to.

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 as a trust-anchor. 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].

4.2.4. Stand-by keys

Stand-by keys are keys that are published in your zone, but are not used to sign RRsets. There are two reasons why someone would want to use stand-by keys. One is to speed up the emergency key rollover. The other is to recover from a disaster that leaves your production private keys inaccessible.

The way to deal with stand-by keys differs for ZSKs and KSKs. To make a stand-by ZSK, you need to publish its DNSKEY RR. To make a stand-by KSK, you need to get its DS RR published at the parent.

Assuming you have your DNS operation at location A, to prepare stand-by keys you need to:

- Generate a stand-by ZSK and KSK. Store them safely in a different location (B) than the currently used ZSK and KSK (that are at location A).
- Pre-publish DNSKEY RR of the stand-by ZSK in the zone.
- Pre-publish DS of the stand-by KSK in the parent zone.

Now suppose a disaster occurs at location A, that disables the access to your currently used keys. To recover from that situation, follow these procedures:

- Set up your DNS operations and import the stand-by keys from location B.
- Post-publish the old ZSK and sign the zone with the stand-by keys.
- After some time, when the new signatures have been propagated, the old ZSK and DS can be removed.
- Generate a new stand-by keyset at a different location and continue "normal" operation.

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 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.

With some type of key rollovers, the DNSKEY is not pre-published and a DNSKEY query tool is not able to retrieve the successor key. In this case, the out-of-band method is required. This also allows the child to determine the digest algorithm of the DS record.

4.3.2. Storing Keys or Hashes?

When designing a registry system one should consider whether to store the DNSKEYs and/or the corresponding DSes. 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-trust-anchor [26]).

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.

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.

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 policy/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 reduces the damage of a successful replay attack. It does mean 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 registry-registrar-registrant 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.

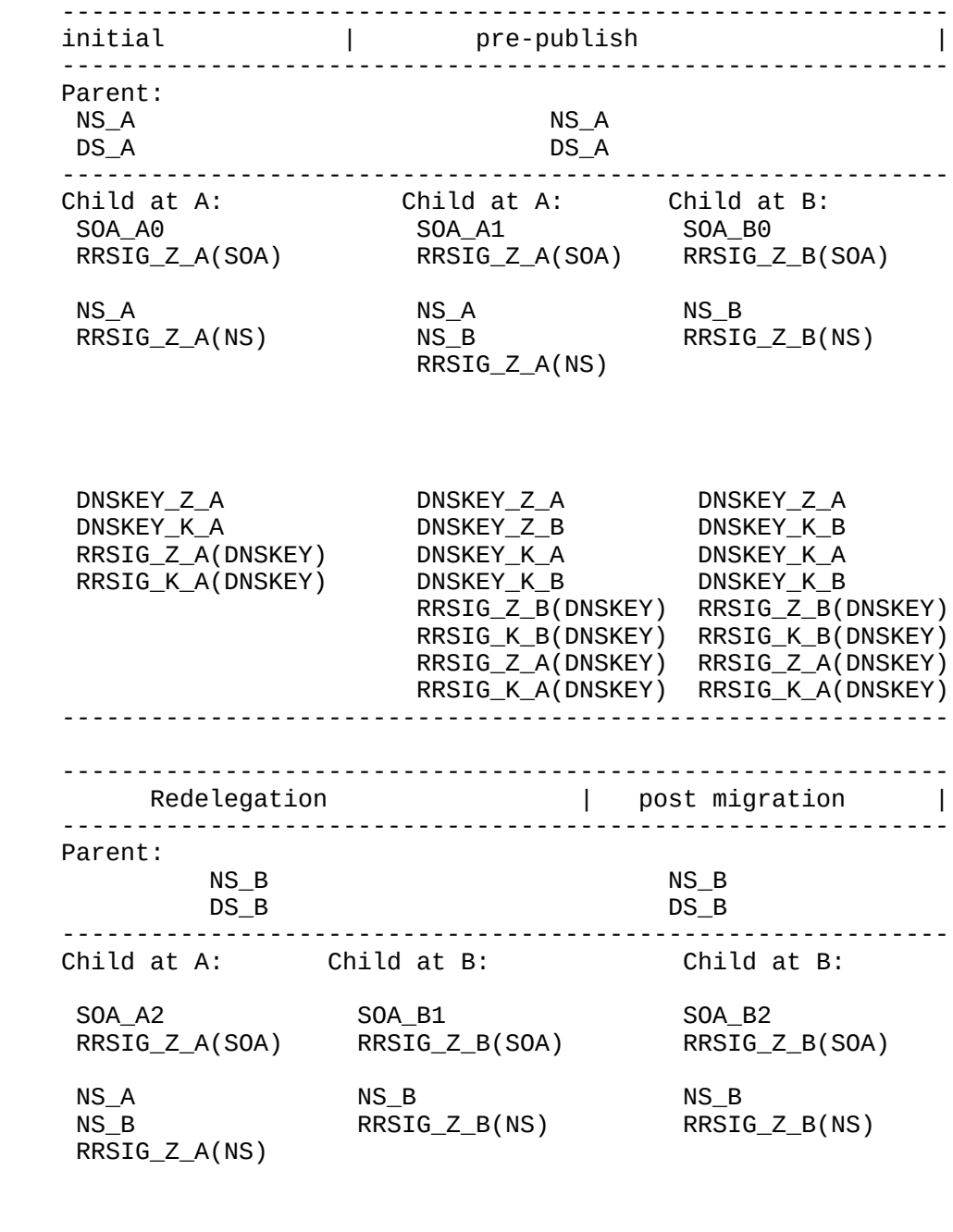
For various reasons, a registrant may want to move between DNS operators. How easy this move will be depends principally on the DNS operator from which the registrant is moving (the losing operator), as they have control over the DNS zone and its keys. The following sections describe the two cases: where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

4.3.5.1. Cooperating 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.



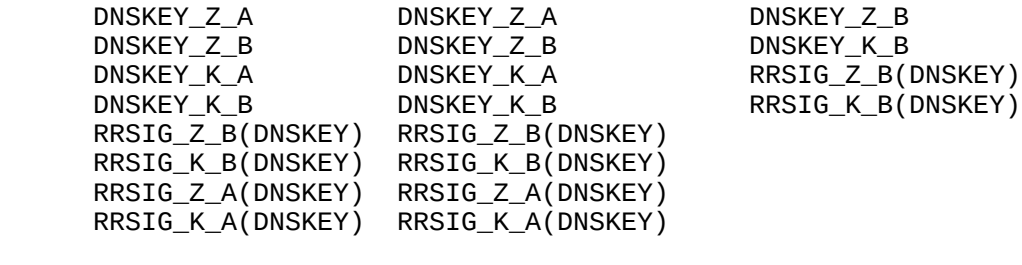


Figure 9: Rollover for cooperating operators

In this figure A denotes the losing operator and B the gaining operator. RRSIG_Z is the RRSIG produced by a ZSK, RRSIG_K 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.

If the registry and registrars allow for DS records to be published, that do not point to a published DNSKEY in the child zone, the Double-DS KSK Rollover is preferred (also known as Pre-Publication KSK Rollover, see Figure 5). This does not require to share the KSK signatures between the operators.

4.3.5.2. Non Cooperating DNS Operators

If the registry and registrars allow for DS records to be published, that do not point to a published DNSKEY in the child zone, the Double-DS KSK Rollover is preferred to resolve the non-cooperative case. The gaining operator publishes a version of the zone, signed with its own key material, and makes a request to the registry to add the corresponding DS record. After the new DS RRset has been propagated to resolver caches, the registrant then asks the registry to remove the DS RR pointing to the losing operator’s DNSKEY.

If Double-DS KSK Rollover is not feasable, things are more complicated, assuming that the losing operator will 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 losing 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.

Note that some behavior of resolver implementations may aid in the process of changing DNS operators:

- TTL sanity checking, as described in RFC2308 [9], will limit the impact the actions of an obstructive, losing operator. Resolvers that implement TTL sanity checking will use an upper limit for TTLs on RRsets in responses.

- If RRsets at the zone cut (are about to) expire, the resolver restarts its search above the zone cut. Otherwise, the resolver risks to keep using a nameserver that might be undelegated by the parent.

- Limiting the time DNSKEYS that seem to be unable to validate signatures are cached and/or trying to recover from cases where DNSKEYs do not seem to be able to validate data, also reduces the effects of the problem of non-cooperating registars.

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:

- 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 RRSet 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.

- We suggest the signature publication period to end at least one Maximum Zone TTL duration (but preferably a few days) 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.

In case of an operational error, you would have one Maximum Zone TTL duration to resolve the problem. Re-signing a zone a few days before the end of the signature validity period ensures the signatures will survive a weekend in case of such operational havoc. This is called the Refresh period (see Section 4.4.2).

- 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 RRSet 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. Data at the leafs in the DNS tree has less impact on recursive nameservers.

- 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 Refresh period (see Section 4.4.2).

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.

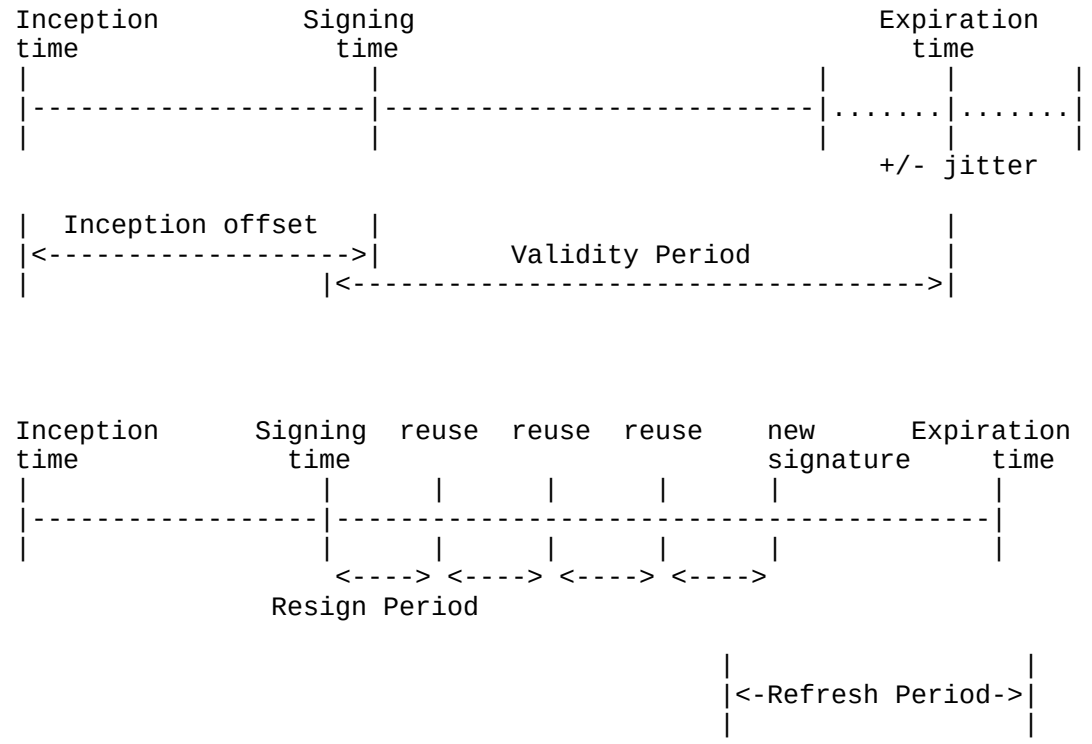


Figure 10: Signature Timing Parameters

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. The inception offset should be chosen so that you minimize the false negatives to a reasonable level.

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.

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:

- includes a list of all the RRTYPEs present which can be used to prove the non-existence of RRTYPEs at a certain name;
- stores only the name for which the zone is authoritative (that is, glue in the zone is omitted); and
- 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 behavior 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 e164.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 a value of 100 iterations is deemed to be a sufficiently costly yet not excessive value: In the worst case scenario, the performance of your nameservers would be halved, regardless of key size [27].

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 usually requires all RRSIG's to be regenerated (you can have a smooth ZSK rollover by honoring the Refresh period). If there is no critical dependency on incremental signing and the 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 Guette, 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 whose 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 thorougly reviewed and provided concrete improvements on the specific types of keyrollovers (e.g. he provided the tables in Appendix C); and

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

Rickard Bellgrim reviewed the document extensively.

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.

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-01 (work in progress), October 2010.

[25] Ljunggren, F., Eklund-Lowinder, A., and T. Okubo, "DNSSEC Policy & Practice Statement Framework", draft-ietf-dnsop-dnssec-dps-framework-03 (work in progress), November 2010.

[26] Larson, M. and O. Gudmundsson, "DNSSEC Trust Anchor Configuration and Maintenance", draft-ietf-dnsop-dnssec-trust-anchor-04 (work in progress), October 2010.

[27] Schaeffer, Y., "NSEC3 Hash Performance", NLnet Labs document 2010-02, March 2010.

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 RRSet in DNSSEC is marked "Bogus" when a signature of an RRSet 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 for RSA keys. It is mathematically more correct to use modulus size for RSA keys, 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 period before the expiration time of the signature, during which the signature is refreshed by the signer.

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 validty 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 RRSet). 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 x 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 'y' denotes a number or an identifier, y 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. A 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 "SOA_x"

SOA representation: SOAs are represented as SOA_x, 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
                           ...
                           BcQ1o99vwn+IS4+J1g== )
       300     NSEC     foo.example.com. NS SOA TXT RRSIG NSEC DNSKEY
```

```
          300    RRSIG     NSEC 5 2 300 20120824013000 (
                           20100424013000 14 example.com.
                           JtHm8ta0diCWYGu/TdrE1O1sYSHblN2i/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+NBIlsJ9mdj5U
                           ...
                           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.
                           xHr023P79YrSHHMtSL0a1nlfUt4ywn/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:

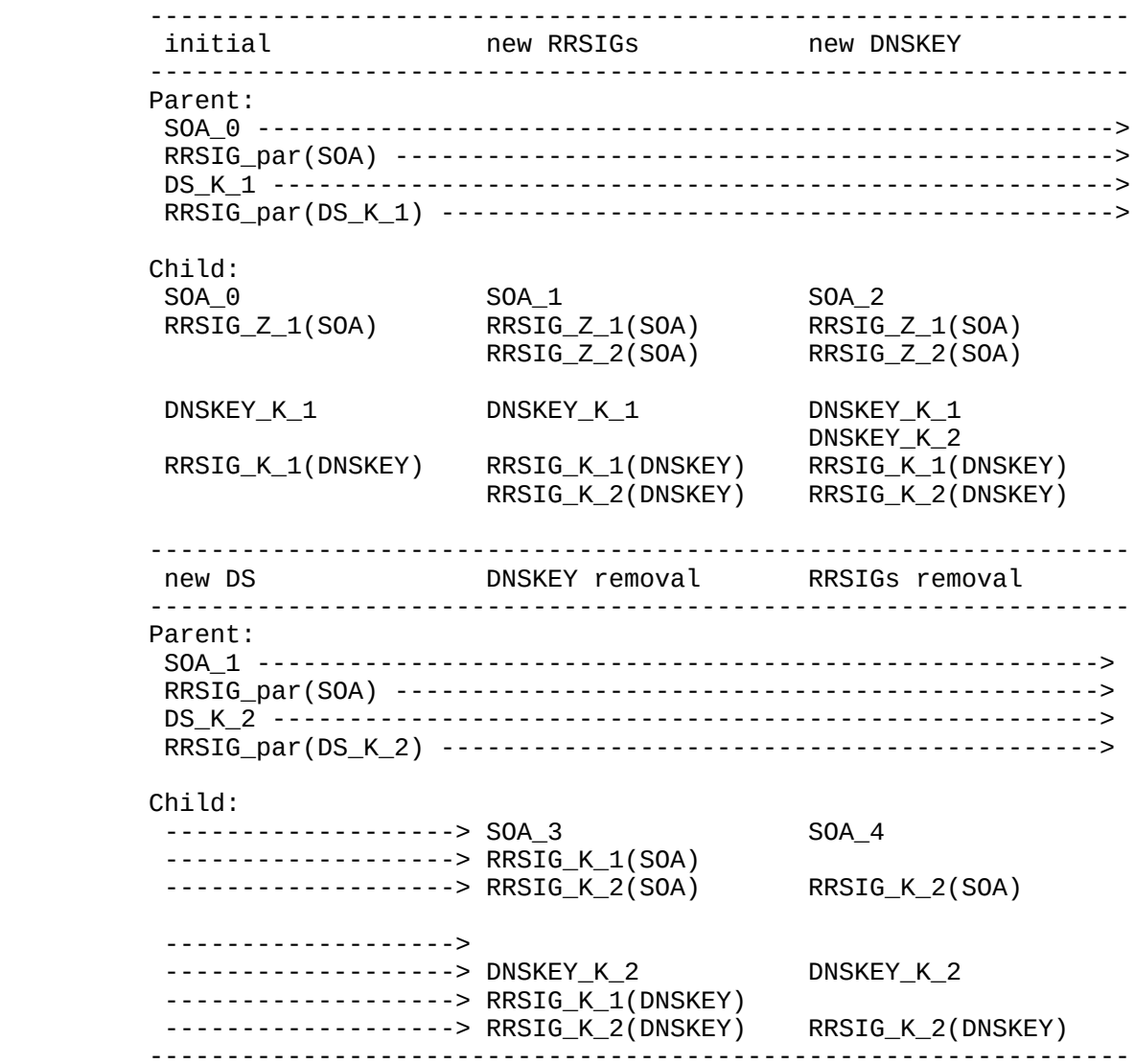
```
SOA_2005092303
RRSIG_Z_14(SOA_2005092303)
DNSKEY_K_14
DNSKEY_Z_15
RRSIG_K_14(DNSKEY)
RRSIG_Z_15(DNSKEY)
```

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

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

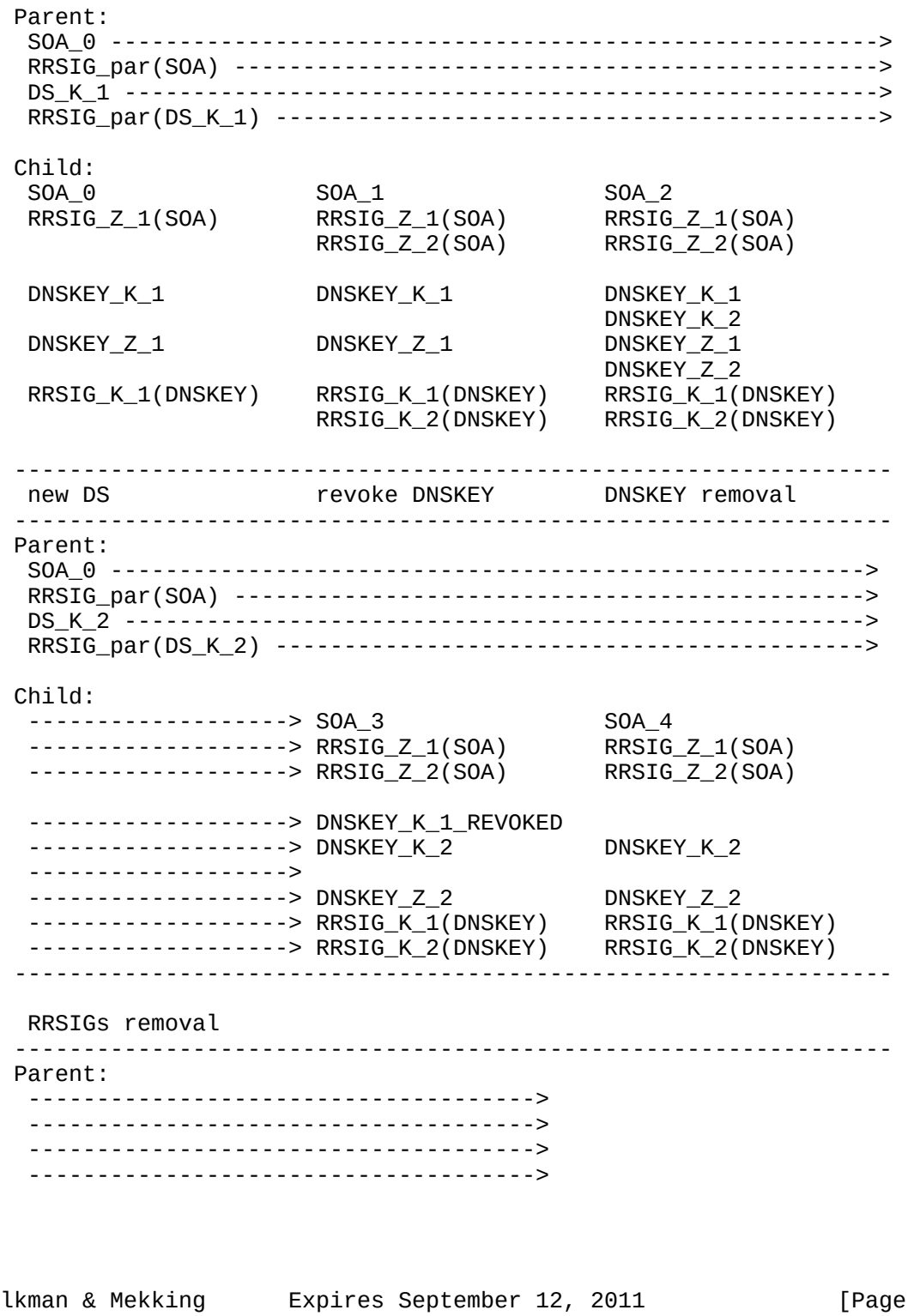
Appendix C. Transition Figures for Special Case Algorithm Rollovers

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



Also see Section 4.1.5.1.

Figure 11: Single Type Signing Scheme Algorithm Roll

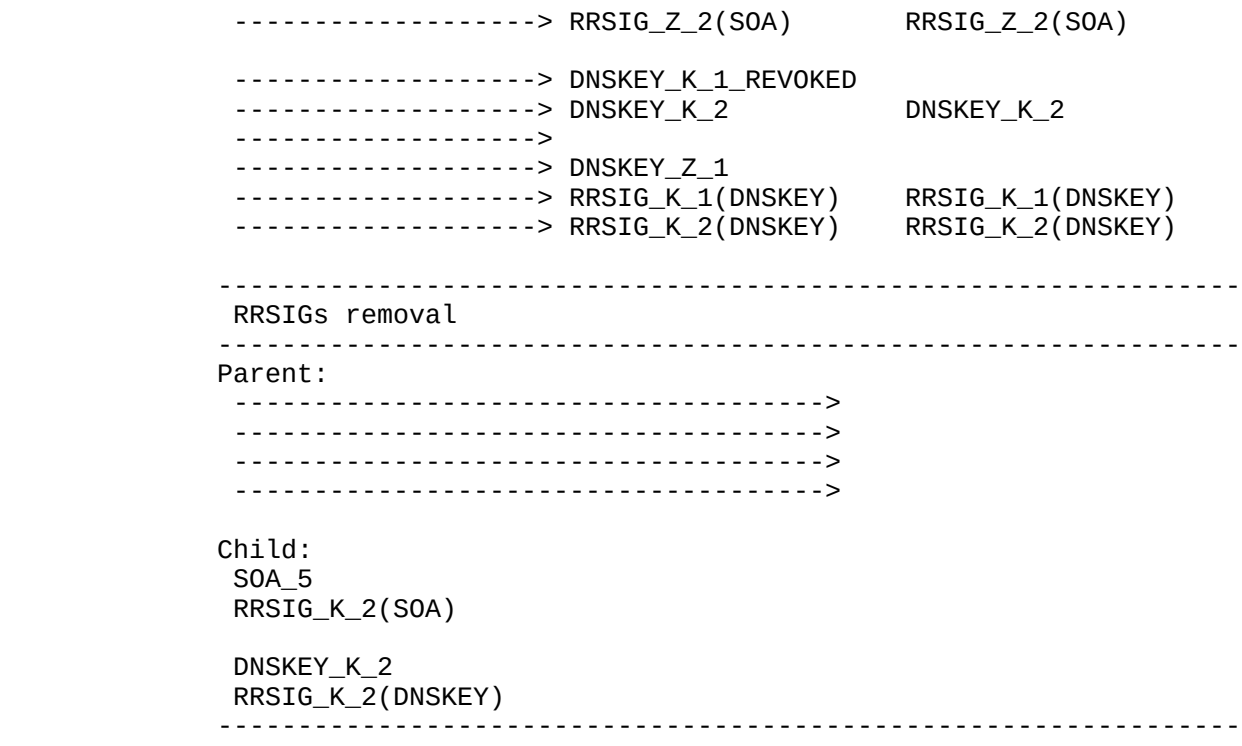

Parent:
SOA_0
RRSIG_par(SOA)
DS_K_1
RRSIG_par(DS_K_1)
Child:
SOA_0
RRSIG_Z_1(SOA)
DNSKEY_K_1
DNSKEY_Z_1
RRSIG_K_1(DNSKEY)
SOA_1
RRSIG_Z_1(SOA)
RRSIG_Z_2(SOA)
DNSKEY_K_1
DNSKEY_Z_1
RRSIG_K_1(DNSKEY)
RRSIG_K_2(DNSKEY)
SOA_2
RRSIG_Z_1(SOA)
RRSIG_Z_2(SOA)
DNSKEY_K_1
DNSKEY_K_2
DNSKEY_Z_1
DNSKEY_Z_2
RRSIG_K_1(DNSKEY)
RRSIG_K_2(DNSKEY)
new DS
revoke DNSKEY
DNSKEY removal
Parent:
SOA_0
RRSIG_par(SOA)
DS_K_2
RRSIG_par(DS_K_2)
Child:
SOA_3
RRSIG_Z_1(SOA)
RRSIG_Z_2(SOA)
DNSKEY_K_1_REVOKED
DNSKEY_K_2
DNSKEY_Z_2
RRSIG_K_1(DNSKEY)
RRSIG_K_2(DNSKEY)
SOA_4
RRSIG_Z_1(SOA)
RRSIG_Z_2(SOA)
DNSKEY_K_2
DNSKEY_Z_2
RRSIG_K_1(DNSKEY)
RRSIG_K_2(DNSKEY)
RRSIGs removal
Parent:

```
   Child:
    SOA_5
    RRSIG_Z_2(SOA)

    DNSKEY_K_2
    DNSKEY_Z_2
    RRSIG_K_2(DNSKEY)
   ------------------------------------------------------------------
```

Also see Section 4.1.5.2.

Figure 12: RFC5011 Style algorithm roll

```
   ------------------------------------------------------------------
    initial              new RRSIGs           new DNSKEY
   ------------------------------------------------------------------
   Parent:
    SOA_0 ------------------------------------------------------------>
    RRSIG_par(SOA) ---------------------------------------------------->
    DS_K_1 ------------------------------------------------------------>
    RRSIG_par(DS_K_1) ------------------------------------------------->

   Child:
    SOA_0                SOA_1                SOA_2
    RRSIG_K_1(SOA)       RRSIG_Z_1(SOA)       RRSIG_Z_1(SOA)
                         RRSIG_Z_2(SOA)       RRSIG_K_2(SOA)

    DNSKEY_K_1           DNSKEY_K_1           DNSKEY_K_1
                                              DNSKEY_K_2
                         DNSKEY_Z_1           DNSKEY_Z_1

    RRSIG_K_1(DNSKEY)    RRSIG_K_1(DNSKEY)    RRSIG_K_1(DNSKEY)
                         RRSIG_K_2(DNSKEY)    RRSIG_K_2(DNSKEY)

   ------------------------------------------------------------------
    new DS               revoke DNSKEY        DNSKEY removal
   ------------------------------------------------------------------
   Parent:
    SOA_0 ------------------------------------------------------------>
    RRSIG_par(SOA) ---------------------------------------------------->
    DS_K_2 ------------------------------------------------------------>
    RRSIG_par(DS_K_2) ------------------------------------------------->

   Child:
    ------------------> SOA_3                SOA_4
    ------------------> RRSIG_Z_1(SOA)       RRSIG_Z_1(SOA)
```



Also see Section 4.1.5.3.

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.

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

- References slightly reordered.
- Applied the errata from http://www.rfc-editor.org/errata_search.php?rfc=4641
- Inserted trivial "IANA considerations" section.

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

D.2. version 0->1

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

- Reference to NIST 800-90 added
- RSA/SHA256 is being recommended in addition to RSA/SHA1.
- 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.
- Replaced the reference to Schneiers' applied cryptography with a reference to RFC4949.
- 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 Verschuren http://www.nlnetlabs.nl/svn/rfc4641bis/trunk/open-issues/non-cooperative-registrars

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

D.3. version 1->2

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

- Added Section 5 based on http://www.nlnetlabs.nl/svn/rfc4641bis/trunk/open-issues/NSEC-NSEC3

- Added a reference to draft-morris-dnsop-dnssec-key-timing [24] for the quantitative analysis on keyrolls

- 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.

- Updated a number of references.

D.4. version 2->3

- Added bulleted list to serve as an introduction on the decision tree in Section 3.

- In section Section 3.1:

  * tried to motivate that key length 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

- 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.

- Added some terms based on http://www.nlnetlabs.nl/svn/rfc4641bis/trunk/open-issues/timing_terminology

- Added paragraph 2 and clarified the second but last paragraph of Section 3.2.2.

- 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.

- Added a paragraph about rolling KSKs that are also configured as trust-anchors in Section 4.1.2
- Added Section 4.1.4.
- Added Section 4.4.2 to address issue "Signature_validity"

D.5. version 3->4

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

D.6. version 4->5

- Improved consistency of notation
- 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. version 5->6

- More improved consistency of notation and some other nits
- Review of Rickard Bellgrim
- Review of Sebastian Castro
- Added a section about Stand-by keys
- Algorithm rollover: Conservative or Liberal Approach
- Added a reference to NSEC3 hash performance report

D.8. Subversion information

www.nlnetlabs.nl/svn/rfc4641bis/

$Id: draft-ietf-dnsop-rfc4641bis.xml 93 2011-03-03 15:16:38Z matje $

Authors' Addresses

Olaf M. Kolkman
NLnet Labs
Science Park 140
Amsterdam 1098 XG
The Netherlands

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

Matthijs Mekking
NLnet Labs
Science Park 140
Amsterdam 1098 XG
The Netherlands

EMail: matthijs@nlnetlabs.nl
URI: http://www.nlnetlabs.nl

Network Working Group
Internet-Draft
Intended status: BCP
Expires: August 4, 2011

J. Abley
D. Knight
ICANN

January 31, 2011

Establishing an Appropriate Root Zone DNSSEC Trust Anchor at Startup
draft-jabley-dnsop-validator-bootstrap-00

Abstract

Domain Name System Security Extensions (DNSSEC) allow cryptographic signatures to be used to validate responses received from the Domain Name System (DNS). A DNS client which validates such signatures is known as a validator.

The choice of appropriate root zone trust anchor for a validator is expected to vary over time as the corresponding cryptographic keys used in DNSSEC are changed.

This document provides guidance on how validators might determine an appropriate trust anchor for the root zone to use at start-up, or when other mechanisms intended to allow key rollover to be tolerated gracefully are not available.

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 August 4, 2011.

Table of Contents

1. Definitions

The terms Key Signing Key (KSK) and Trust Anchor are used as defined in [RFC4033].

The term Validator is used in this document to mean a Validating Security-Aware Stub Resolver, as defined in [RFC4033].

2. Introduction

The Domain Name System (DNS) is described in [RFC1034] and [RFC1035]. DNS Security Extensions (DNSSEC) are described in [RFC4033], [RFC4034] and [RFC4035].

The root zone of the DNS was signed using DNSSEC in July 2011, and many top-level domain registries have since signed their zones, installing secure delegations for them in the root zone. A single trust anchor for the root zone is hence increasingly sufficient for validators.

Validators are deployed in a variety of environments, and there is variation in the amount of system administration that might reasonably be expected to be available. For example, embedded devices might never be administered by a human operator, whereas validators deployed on general-purpose operating systems in enterprise networks might have technical staff available to assist with their configuration.

This document includes descriptions of mechanisms for validator bootstrapping, intended to be sufficient for embedded devices. The implementation of those mechanisms might be automatic in the case of unattended devices, or manual, carried out by a systems administrator, depending on local circumstances.

The choice of appropriate trust anchor for a DNSSEC Validator is expected to vary over time as the corresponding KSK used in the root zone is changed. The DNSSEC Policy and Practice Statement (DPS) for the root zone KSK maintainer [KSK-DPS] specifies that scheduled KSK rollover will be undertaken according to the semantics specified in [RFC5011]. Validators which are able to recognise and accommodate those semantics should need no additional support to be able to maintain an appropriate trust anchor over a root zone KSK rollover event.

The possibility remains, however, that [RFC5011] signalling will not be available to a validator: e.g. certain classes of emergency KSK rollover may require a compromised KSK to be discarded more quickly than [RFC5011] specifies, or a validator might be off-line over the whole key-roll event.

This document provides guidance on how DNSSEC Validators might determine an appropriate set of trust anchors to use at start-up, or when other mechanisms intended to allow key rollover to be tolerated gracefully are not available.

The bootstrapping procedures described in this document are also

expected to be useful for a deployed, running validator which is not able to accommodate a KSK roll using [RFC5011] signalling.

3. Summary of Approach

A validator that has no valid trust anchor initialises itself as follows.

3.1. Initial State

A validator in its initial state is capable of sending and receiving DNS queries and responses, but is not capable of validating signatures received in responses.

A validator must confirm that its local clock is sufficiently accurate before trust anchors can be established, and before processing of DNSSEC signatures can proceed. Discussion of timing considerations can be found in Section 4.

3.2. Trust Anchor Retrieval

Once the local clock has been synchronised, a validator may proceed to gather candidate trust anchors for consideration. Discussion of trust anchor retrieval can be found in Section 5.

3.3. Trust Anchor Selection

Once a set of candidate trust anchors has been obtained, a validator attempts to find one trust anchor in the set which is appropriate for use. This process involves verification of cryptographic signatures, and is discussed in Section 6.

3.4. Full Operation

The validator now has an accurate trust anchor for the root zone, and is capable of validating signatures on responses from the DNS.

4. Timing Considerations

DNSSEC signatures are valid for particular periods of time, as specified by the administrator of the zone containing the signatures. It follows that any validator must maintain an accurate local clock in order to verify that signatures are accurate.

Trust anchors correspond to KSKs in particular zones. Zone administrators may choose to replace KSKs from time to time, e.g. due to a key compromise or local key management policy, and the corresponding appropriate choice in trust anchor will change as KSKs are replaced.

Trust anchors for the root zone in particular are published with intended validity periods, as discussed in Section 5. A validator making use of such trust anchors also requires an accurate local clock in order to avoid configuring a local trust anchor which corresponds to an old key.

Validators should take appropriate steps to ensure that their local clocks are set with sufficient accuracy, and in the case where local clocks are set with reference to external time sources over a network [RFC5905] that the time information received from those sources is authentic.

5. Retrieval of Candidate Trust Anchors

Candidate trust anchors may be retrieved using several mechanisms. The process of gaining trust in particular candidate trust anchors before using them is discussed in Section 6.

5.1. Retrieval of Trust Anchors from Local Sources

A trust anchor which is packaged with validator software can never be trusted, since the corresponding root zone KSK may have rolled since the software was packaged, and the trust anchor may be derived from a root zone KSK that was retired due to compromise.

Validators should never use local trust anchors for bootstrapping.

5.2. Retrieval of Trust Anchors from the DNS

The current root zone trust anchor is a hash (in DS RDATA format) of a member of the root zone apex DNSKEY RRSet that has the SEP bit set. Such a trust anchor could be derived from a response to the query ". IN DNSKEY?", but there is no mechanism available to trust the result: without an existing, accurate trust anchor the validator has no means to gauge the authenticity of the response.

Validators should never derive trust anchors from DNSKEY RRSets obtained from the DNS.

5.3. Retrieval of Trust Anchors from the Root Zone KSK Manager

The Root Zone KSK Manager publishes trust anchors corresponding to the root zone KSK as described in [I-D.jabley-dnssec-trust-anchor].

A full history of previously-published trust anchors, including the trust anchor recommended for immediate use, is made available in an XML document at the following stable URLs:

- <http://data.iana.org/root-anchors/root-anchors.xml>
- <https://data.iana.org/root-anchors/root-anchors.xml>

Validity periods for each trust anchor packaged in the root-anchors.xml document are provided as XML attributes, allowing an appropriate trust anchor for immediate use to be identified (but see Section 4).

Individual trust anchors are also packaged as X.509 identity certificates, signed by various Certificate Authorities (CAs). URLs to allow those certificates to be retrieved are included as optional

elements in the XML document.

For automatic bootstrapping, the recommended approach is as follows.

1. Retrieve <http://data.iana.org/root-anchors/root-anchors.xml>

2. Identify the trust anchors which are valid for current use, with reference to the current time and date.

3. Retrieve the corresponding X.509 identity certificates for the key identified in the previous step, for use in establishing trust in the retrieved trust anchor (see Section 6).

6. Establishing Trust in Candidate Trust Anchors

Once a candidate trust anchor has been retrieved, the validator must establish that it is authentic before it can be used. This document recommends that this be carried out by checking the signatures on each of the X.509 identity certificates retrieved in the previous step until a certificate is found which matches a CA trust anchor.

This verification phase requires that validators ship with a useful set of CA trust anchors, and that corresponding identity certificates are published by the root zone KSK manager. In some cases validator implementors may decide to use commercial CA services, perhaps a subset of the "browser list" that is commonly distributed with web browsers; alternatively a vendor may instantiate its own CA and make arrangements with the root zone KSK manager to have the corresponding identity certificate locations published in root-anchors.xml.

The CA trust anchors packaged with validators should have an expected lifetime in excess of the anticipated life of the validator. As a protection against CA failure, validators are recommended to ship with more than one CA trust anchor.

7. Failure to Locate a Valid Trust Anchor

A validator that has failed to locate a valid trust anchor may re-try the retrieval and trust establishment phases indefinitely, but must not perform validation on DNS responses until a valid trust anchor has been identified.

8. IANA Considerations

This document has no IANA actions.

9. Security Considerations

This document discusses an approach for automatic configuration of trust anchors in a DNSSEC validator.

10. Normative References

[I-D.jabley-dnssec-trust-anchor]
Abley, J. and J. Schlyter, "DNSSEC Trust Anchor Publication for the Root Zone", draft-jabley-dnssec-trust-anchor-01 (work in progress), October 2010.

[KSK-DPS] Ljunggren, F., Okubo, T., Lamb, R., and J. Schlyter, "DNSSEC Practice Statement for the Root Zone KSK Operator", May 2010.

[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.

[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.

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

Appendix A. Acknowledgements

This document contains material first discussed at VeriSign and ICANN during the deployment of DNSSEC in the root zone, and also draws upon subsequent technical discussion from public mailing lists. The contributions of all those who voiced opinions are acknowledged.

Appendix B. Editorial Notes

This section (and sub-sections) to be removed prior to publication.

B.1. Discussion

This is not a working group document. However, the topics discussed in this document are consistent with the general subject area of the DNSOP working group, and discussion of this document could reasonably take place on the corresponding mailing list.

B.2. Change History

00 Initial draft.

Authors' Addresses

Joe Abley
ICANN
4676 Admiralty Way, Suite 330
Marina del Rey, CA 90292
USA

Phone: +1 519 670 9327
Email: joe.abley@icann.org

Dave Knight
ICANN
4676 Admiralty Way, Suite 330
Marina del Rey, CA 90292
USA

Phone: +1 310 913 4512
Email: dave.knight@icann.org

Domain Name System Operations
Internet-Draft
Intended status: Standards Track
Expires: August 29, 2011

W. Mekking
NLnet Labs

February 25, 2011

DNSSEC Key Timing Considerations Follow-Up
draft-mekking-dnsop-dnssec-key-timing-bis-00

Abstract

This document describes issues surrounding the timing of events in enforcing key policy within DNSSEC. It presents timelines for various key rollovers and changes into the policy with respect to the key signing scheme. It 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 August 29, 2011.

Copyright Notice

Table of Contents

1. Introduction

A zone is managed according to a given security policy. Such a policy may enforce DNSSEC keys to be used and for how long. When enforcing a lifetime on DNSSEC keys, key rollovers must to take place. In addition, changes in the policy may trigger certain key rollover events. Key rollovers are time critical, multiple steps processes. This document describes issues surrounding the timing of events in the rolling of DNSSEC keys.

[MM: Editorarial comments are indicated by square brackets and editor initials]

1.1. Key Rollover Considerations

A key is used with a purpose: An operational decision has been made to secure the zone with DNSSEC. That decision leads to a key being created, published in the zone and used for signing. Policy may enforce a lifetime on keys. As a result, current active keys need to be replaced with new keys. The new key becomes active, while the current key is retired. The keys need to be introcuded into and removed from the zone at the appropriate times. Considerations that must be taken into account are:

- Speed: A rollover should occur as fast and simple as possible. However, DNSSEC records are not only held at the authoritative nameserver, they are also cached at client resolvers. The data on these systems can be interlinked, meaning a validating may try to validate a signature retrieved from a cache with a key obtained separately. The rollover process needs to happen in such a way that at all times through the rollover the information is consistent.

- Size of the zone and the DNS response: A rollover can be speed up by introducing the DNSSEC records prematurely. However, adding arbitrary signatures increases the size of your zone and DNS responses significantly. To keep the sizes of the zone and responses as small as possible, you might want to consider to introduce the DNSSEC records only when they are required, For the same reason, dead keys and signatures must be removed periodically.

- Size of the DNSKEY RRset and the priming response: You can choose to keep the size of the DNSKEY RRset to a minimum, to make priming responses smaller in size. The larger the packet, the more resolvers may have problems retrieving the response. Other responses may have more signatures, since the initial size is relatively small. The DNSKEY RRset is usually already quite large

and should not grow too much anymore.

- Interactions with the Parent: A KSK sometimes needs its corresponding DS record to be published at the parent zone, while its predecessor needs to remove its DS record from the parent zone. Such a request requires additional operational work and can be a sufficient delay. Ideally, the interactions with the parent is kept to a minimum.

1.1.1. Key Goals

We have identified three different goals for a key:

- Activate key: Make validating resolvers use the key's associated information to perform authentication.
- Remove key: Make validating resolvers forget about the key's associated information.
- Stand-by key: Pre-publish information for this key to speed up a future (unscheduled) rollover.

Each key rollover and change in key signing scheme can now be described by one or more goals that are put on a key.

1.2. Terminology

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

2. Key Definitions

2.1. Key Types

Keys can be used to authenticate information within the zone. Such keys are said to be ZSKs. In addition, keys can be used to authenticate the DNSKEY RRset in the zone. These keys are said to be KSKs. Keys can be marked to be ZSK and KSK at the same time, for example in a Single Type Signing Scheme (STSS).

Despite that ZSK and KSK only describe the usage of a key, the terms are often used for identifiying a key. Thus when we talk about a ZSK, we actually mean that the key is used as ZSK. In the same spirt, a KSK is a key that is used as KSK.

DNSSEC recognises the classification of keys with its SEP bit set and not set. Usually if a key is used as KSK, the SEP bit is set. However, draft-ietf-dnsext-dnssec-bis-updates [dnssec-bis] says that

a SEP bit setting has no effect on how a DNSKEY may be used. Policy determines whether the bit should be set, depending on the key's usage.

2.2. Key States Unraveled

We use unraveled key states to seperately represent the key and its associated information. There can be up to three pieces of key associated information: the public key (in DNSKEY format), its created signatures (the RRSIG records) and the secured delegation (the corresponding DS record). The state of the piece of information is defined by 'RRtype State'.

Key conditions are essentially what are called key states in draft-ietf-dnsop-dnssec-key-timing [key-timing]. A key can have multiple conditions at the same time.

A piece of information may exist in up to two places: it can be present in the corresponding zone and it may live in resolver caches. This is true for every piece of associated information. Therefore, all of the three pieces of information follow the same state diagram:

Generated --> Introduced --> Propagated --> Withdrawn --> Dead.

Generated: The information has been generated, but is not available in the zone. In this state, no resolvers are able to fetch this information.
- The key condition is said to be Generated, if no information has passed the Introduced state yet.

Introduced: The information is introduced and, as a result, may be available in the zone. In this state, there may be resolvers that fetch this information.
- The key condition is said to be Published if it has its DNSKEY state in Introduced.
- The key condition is said to be Active if it has its RRSIG state in Introduced (for ZSKs).
- The key condition is said to be Submitted, or ActiveDS, if it has its DS state in Introduced (for KSKs).

Propagated: The information is available in the zone and enough time has passed to have it propagated into all resolver caches. As a result, all resolvers fetch this information from cache of from the authoritative name server.
- The key condition is said to be Known if it has its DNSKEY state in Propagated.
- The key condition is said to be Safe if it has its RRSIG state in Propagated (for ZSKs).

- The key condition is said to be SafeDS if it has its DS state in Propagated (for KSKs).

Withdrawn: The information is being withdrawn from the zone, but may still be available in the zone. In this state, the information can still live in resolver caches.
- The key condition is said to be Removed if it has its DNSKEY state in Withdrawn.
- The key condition is said to be Retired if it has its RRSIG state in Withdrawn (for ZSKs).
- The key condition is said to be RetiredDS if it has its DS state in Withdrawn (for KSKs).

Dead: The information is not available in the zone anymore and enough time has passed to have it expire from all resolver caches.
- The key condition is said to be Forgotten if it has its DNSKEY state in Dead.
- The key condition is said to be Expired if it has its RRSIG state in Dead (for ZSKs).
- The key condition is said to be ExpiredDS if it has its DS state in Dead (for KSKs).

A key state can now be represented as the triplet (DNSKEY State, RRSIG State, DS State). For example:

S(Kc) = (DNSKEY Propagated, RRSIG Introduced, DS Generated)

tells us that key Kc is published in the zone and all the resolvers that have a copy of the DNSKEY RRset, have one that includes Kc. In other words, Kc is said to be Known. In addition, the key is Active as it is being used for signing RRsets: RRSIG records made with Kc have been introduced in the zone. However, there may still be some resolver caches that are unaware of these signatures. Finally, the corresponding DS record is said to be Generated and has thus not yet been submitted to the parent.

For convenience, we can represent a ZSK as a tuple (DNSKEY State, RRSIG State), because the DS record is only used with KSKs. And we can represent a KSK as a tuple (DNSKEY State, DS State), because the RRSIG state only refers to ZSKs. The RRSIG record over the DNSKEY RRset should be published at the same time when the corresponding DNSKEY record is published. Therefore, both records will propagate and expire at the same time from resolver caches.

2.3. Delay Timings

For every change we make in the zone, we have to take into account several delays.

Software Delay (Dsfw): The time it takes for the software to introduce the new information in the zone. This delay can vary alot depending on the information that needs to be introduced. One can imagine that the software needs more time to sign a complete zone than when it pre-publishes a DNSKEY record. [MM: Dsfw maps to Dsgn from the key-timing draft]

Propagation Delay (Dprp): The time it takes for any change introduced at the master to replicate to all slave servers.

TTL Delay (Dttl): The time it takes to expire the previous information from the resolver caches. This delay depends on what RRsets need to expire from the caches. If not explicitly mentioned otherwise, Dttl is considered the maximum TTL of the information that needs to expire from caches. Otherwise, Dttl(RRtype) shows which specific RRsets need to expire. [MM: TTL terminology in key-timing draft: TTLds, TTLkey, TTLkeyC, TTLsoa, TTLsoaC, TTLsoaP, TTLsig)]

Registration Delay to the Parent (Dreg): The time it takes to get the DS record to be placed into the parent zone, after it is submitted.

Propagation Delay of the Parent (DprpP): The time it takes for any change introduced at the parent master to replicate to all parent slave servers.

Despite these delays may vary for the different rollover methods, we can identify the propagation delay to the caches as:

```
DcacheZ = Dsfw + Dprp + Dttl
DcacheK = Dsfw + Dprp + Dttl(DNSKEY)
DcacheP = Dreg + DprpP + Dttl(DS)
```

where DcacheZ is the propagation delay to the caches for information published in our zone, DcacheK is the propagation delay to the caches for our DNSKEY RRset and DcacheP is the propagation delay for information published in our parent zone.

3. Key Rollovers

There are many different key rollover methods. In Section 1.1, we have seen that there are several properties to prefer one method over the other. Though there are many different type of key rollovers, all methods share the same goal. There is a current key (Kc) that needs to become Forgotten-Retired and a successor key (Ks) that needs to become Known-Safe.

3.1. ZSK Rollovers

The two most common rollover methods for ZSKs are Double-Signature and Pre-Publication. Both are described in RFC4641 [RFC4641]. draft-ietf-dnsop-dnssec-key-timing [key-timing] also introduces ZSK Double-RRSIG rollover. Double-Signature is the fastest way to rollover a ZSK. Pre-Publication minimizes the number of signatures over the RRsets in the zone and responses. Double-RRSIG keeps the size of the DNSKEY RRset to a minimum.

3.1.1. Double-Signature

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, 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.

Double-Signature is the fastest way to rollover to a new key, since all new information is published right away. 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.

Only when Ks is said to be Known, e.g. the DNSKEY record of Ks is known to all validating resolvers, we can remove the signatures made with Kc. And only when we can ensure that all validators only use the information of Ks for authentication, we can remove the DNSKEY record for Kc. In other words, Ks needs to be Known and Safe, before we can remove Kc. Thus, we first have to introduce all new information into the zone. Once all has been propagated, we can withdraw all information of Kc from the zone.

The timeline diagram is shown below:

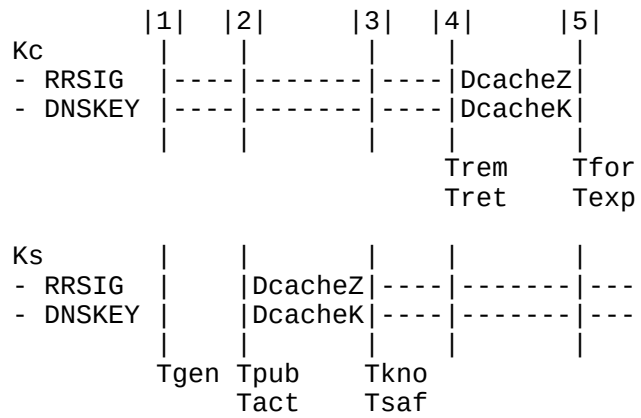


Figure: ZSK Double-Signature Rollover.

Event 1: Key Ks is generated at the generate time (Tgen).

S(Ks) = (DNSKEY Generated, RRSIG Generated)
C(Ks) = Generated

Event 2: Key Ks is added to the DNSKEY RRset and is immediately used to sign the zone; existing signatures in the zone are not removed. This is Ks' publish time (Tpub) and Ks is said to be Published. It is also Ks' active time (Tact), the time when Ks is said to be Active. Because the Double-Signature rollover is in place, we now temporarily have two active keys.

Tpub(Ks) >= Tgen(Ks), Tact(Ks) == Tpub(Ks)

S(Ks) = (DNSKEY Introduced, RRSIG Introduced)
C(ks) = Published Active

Event 3: The information for Ks must be published long enough to ensure that the information have reached all validating resolvers that may have RRsets from this zone cached. At the point in time that the DNSKEY RRset including Ks has been propagated and Ks is said to be Known (Tkno). At the point in time that the other RRsets including a signature of Ks have been propagated (Tsaf), Ks is said to be Safe.

Tkno(Ks) >= Tpub(Ks) + DcacheK
Tsaf(Ks) >= Tact(Ks) + DcacheZ

S(Ks) = (DNSKEY Propagated, RRSIG Propagated)

C(ks) = Known Safe

Note that we could already retire Kc, i.e. stop signing with Kc, after DcacheK. It does not matter if not all signatures of Ks have been Propagated, since the resolver can validate RRsets with both Kc and Ks. If the validator fetches a RRset from the cache, it uses the DNSKEY of Kc for validation. Otherwise, it can use the DNSKEY of Ks.

Event 4: Once we have a successor key that is said to be Propagated, we can retire Kc. This is Kc' retire time (Tret) and Kc is said to be Retired. And once we have a successor key that is said to be Safe, we can remove Kc. Therefore, it is also Kc' removal time (Trem), the time that Kc is said to be Removed.

Tret(Kc) >= Tkno(Ks)
Trem(Kc) >= MAX(Tsaf(Ks), TsafDS(Ks))

S(Kc) = (DNSKEY Withdrawn, RRSIG Withdrawn)
C(kc) = Removed Retired

Event 5: From the perspective of the authoritative server, the rollover is complete. After some delay, Kc and its signatures have expired from the caches. This delay is the maximum of DcacheZ, DcacheK. This is Tfor, the time that the key is said to be Forgotten and Texp, the time that the key is said to be Expired.

Tfor(Kc) >= Trem(Kc) + DcacheK
Texp(Kc) >= Tret(Kc) + DcacheZ

S(Kc) = (DNSKEY Dead, RRSIG Dead)
C(Kc) = Forgotten Expired

3.1.2. Pre-Publication

With Pre-Publication, 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 can be removed. During the re-signing process 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 can be removed from the DNSKEY RRset.

Pre-Publication is more complex than Double-Signature - introduce the new key, approximately one TTL later sign the records, and approximately one TTL after that remove the old key. It also takes more time than the Double-Signature method. The delay is because we don't want to publish signatures of both keys at the same time. As an advantage, the amount of DNSSEC data is kept to a minimum which reduces the impact on performance.

The timeline diagram looks like this:

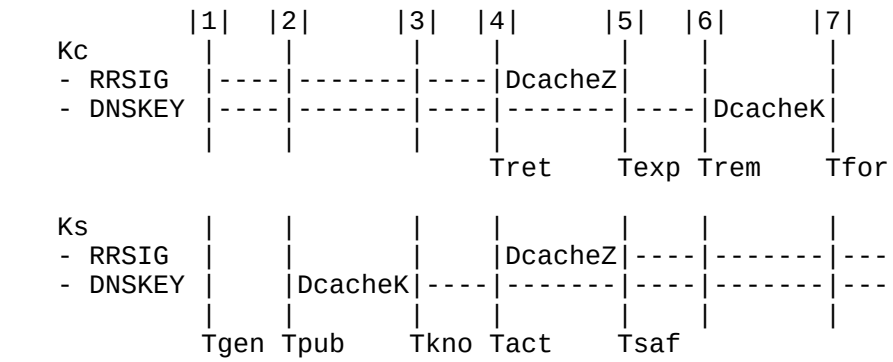


Figure: ZSK Pre-Publication Rollover.

Event 1: Key Ks is generated at the generate time (Tgen).

S(Ks) = (DNSKEY Generated, RRSIG Generated)
C(Ks) = Generated

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

Tpub(Ks) >= Tgen(Ks)

S(Ks) = (DNSKEY Introduced, RRSIG Generated)
C(Ks) = Published

Event 3: Before Ks can be used, the DNSKEY record for Ks must be published for long enough (DcacheK) to guarantee that any resolver that has a copy of the DNSKEY RRset also includes this key. In other words, that any prior cached information about the DNSKEY RRset has

expired. After this delay, the key is said to be Known and could be used to sign records. The time at which this event occurs is Tkno, which is given by:

Tkno(Ks) >= Tpub(Ks) + DcacheK

S(Ks) = (DNSKEY Propagated, RRSIG Generated)
C(Ks) = Known

Event 4: At some point in time, the decision is made to actually 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. This point in time is Ks' active time (Tact), the time that Ks is said to be Active. It is also Kc' retire time (Tret), the time that Kc is said to be Retired.

Tact(Ks) >= Tkno(Ks), Tret(Kc) == Tact(Ks)

S(Kc) = (DNSKEY Propagated, RRSIG Withdrawn)
C(Kc) = Known Retired
S(Ks) = (DNSKEY Propagated, RRSIG Introduced)
C(Ks) = Known Active

Event 5: Kc needs to be retained in the zone whilst any RRSIG records created by the retired key are still published in the zone or held in resolver caches. In other words, Kc should be retained in the zone until all RRSIG records created by Ks have been propagated. This time is Ks' safe time (Tsaf), the time that Ks is considered to be Safe. Consequently, at the same time Kc is considered to be Expired.

Tsaf(Ks) >= Tact(Ks) + DcacheZ

S(Kc) = (DNSKEY Propagated, RRSIG Dead)
C(Kc) = Known Expired
S(Ks) = (DNSKEY Propagated, RRSIG Propagated)
C(Ks) = Known Safe

Event 6: When all new signatures have been propagated, Kc can be removed from the zone and the DNSKEY RRset re-signed with the current KSK. This time is Kc' removal time (Trem), the time that Kc is considered to be Removed.

Trem(Kc) >= Tsaf(Ks)

S(Kc) = (DNSKEY Withdrawn, RRSIG Dead)
C(Kc) = Removed Expired

Event 7: From the perspective of the authoritative server, the

rollover is complete. After some delay, The DNSKEY record for Kc has expired from the caches. This is Tfor, and the key is said to be Forgotten.

Tfor(Kc) >= Trem(Kc) + DcacheK

S(Kc) = (DNSKEY Dead, RRSIG Dead)
C(Kc) = Forgotten Expired

3.1.3. Double-RRSIG

This involves introducing the new signatures first, while existing signatures are being retained. This state of affairs remains in place for long enough to ensure that all RRsets cached in client validating resolvers contain two signatures. The DNSKEY RR can now be switched. For the period of time before the predecessor key has been expired from all caches, it does not matter if the validator uses the cached key or the successor key that is in the zone. Both corresponding signatures can be retrieved from the cache or from the name server.

Once the signing process is complete and enough time has elapsed to allow all old information to expire from caches, the old 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.

Double-RRSIG is also more complex than Double-Signature - first introducing the signatures, then switch the key and finally remove the olds signatures. It also takes more time than the Double-Signature method. The delay is because we cannot publish the public data of both keys at the same time. As an advantage, the DNSKEY RRset is kept to a minimum which reduces the impact on priming performance.

The timeline diagram is shown below:

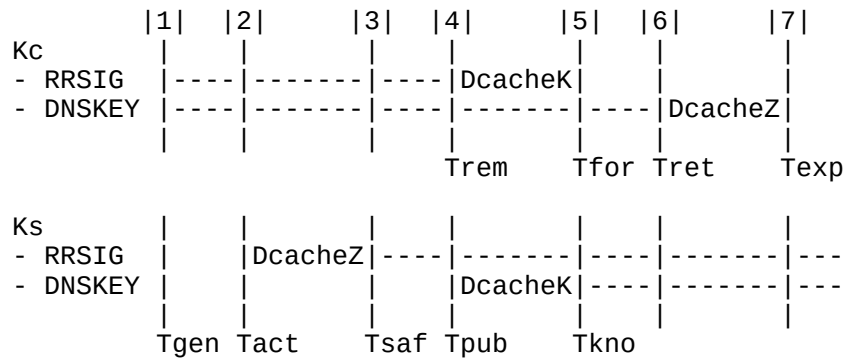


Figure: ZSK Double-RRSIG Rollover.

Event 1: Key Ks is generated at the generate time (Tgen).

S(Ks) = (DNSKEY Generated, RRSIG Generated)
C(Ks) = Generated

Event 2: The zone is signed with Ks but existing signatures are retained. The DNSKEY RR for Ks remains unpublished. The time at which this occurs is Ks' active time (Tact), and the key is now said to be Active.

Tact(Ks) >= Tgen(Ks)

S(Ks) = (DNSKEY Generated, RRSIG Introduced)
C(Ks) = Active

Event 3: Before we can switch the DNSKEY from Kc to Ks, the signatures of Ks must be published for long enough (DcacheZ) to guarantee that any resolver that has a copy of any RRset, also has both signatures. In other words, that any cached information is double signed. After this delay, the key is said to be Safe. The time at which this event occurs is Tsaf, which is given by:

Tsaf(Ks) >= Tact(Ks) + DcacheZ

S(Ks) = (DNSKEY Generated, RRSIG Propagated)
C(Ks) = Safe

Event 4: At some point in time, the decision is made to publish Ks. This point in time is Ks' publish time (Tpub), the time that Ks is said to be Published. At the same time, the DNSKEY RR for Kc is removed from the zone, and Kc is said to be Removed.

Tpub(Ks) >= Tsaf(Ks), Trem(Kc) == Tpub(Ks)

S(Kc) = (DNSKEY Removed, RRSIG Propagated)
C(Kc) = Removed Safe
S(Ks) = (DNSKEY Introduced, RRSIG Propagated)
C(Ks) = Published Safe

Event 5: The signatures of Kc need to be retained in the zone until the DNSKEY RR has expired from all resolver caches. When this happens, Ks is said to be Known (Tkno) and Kc is said to be Forgotten (Tfor).

Tfor(Kc) >= Trem(Kc) + DcacheK
Tkno(Ks) >= Tpub(Ks) + DcacheK

S(Kc) = (DNSKEY Dead, RRSIG Propagated)
C(Kc) = Forgotten Safe
S(Ks) = (DNSKEY Propagated, RRSIG Propagated)
C(Ks) = Known Safe

Event 6: The signatures of Kc can be removed when the DNSKEY RR for Ks has been propagated. This time is Kc' retire time (Tret), the time that Kc is considered to be Retired.

Tret(Kc) >= Tsaf(Ks)

S(Kc) = (DNSKEY Dead, RRSIG Withdrawn)
C(Kc) = Forgotten Retired

Event 7: From the perspective of the authoritative server, the rollover is complete. After some delay, all signatures of Kc have expired from the caches. This is Texp, and the key is said to be Expired.

Texp(Kc) >= Tret(Kc) + DcacheZ

S(Kc) = (DNSKEY Dead, RRSIG Dead)
C(Kc) = Forgotten Expired

3.2. KSK Rollovers

The most common rollover method for KSKs is Double-Signature, described in RFC4641 [RFC4641]. Two more methods are identified in draft-ietf-dnsop-dnssec-key-timing [key-timing]: Double-DS and Double-RRset. Double-RRset is the fastest way to rollover a KSK, while Double-Signature minimizes the number of required interactions to the parent, and Double-DS keeps your DNSKEY RRset as small as possible.

Note that with these type of rollovers, we do not have to worry whether the information within the zone is authentic. We assume that there exists one or more ZSKs in the DNSKEY RRset that takes care of this during the rollover.

3.2.1. Double-RRset

With 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.

Only when Ks is said to be Known, e.g. the DNSKEY record of Ks is known to all validating resolvers, we can remove the DS record of Kc. And only when can ensure that all validators can use the DS record for Ks to build the secure chain of trust, we can remove the DNSKEY record of Kc. In other words, Ks needs to be Known and SafeDS. Thus, we first have to introduce all new information into the zone. Once all has been propagated, we can withdraw all information of Kc from the zone.

The timeline diagram looks like this:

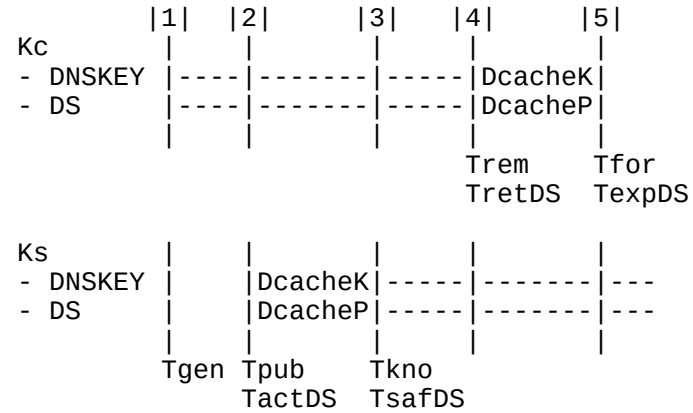


Figure: KSK Double-RRset Rollover.

Event 1: Ks is generated at time Tgen.

S(Ks) = (DNSKEY Generated, DS Generated)
C(Ks) = Generated

Event 2: Ks is introduced into the zone; it is added to the DNSKEY RRset, which is then signed by all currently active KSKs (including

Kc and Ks). In addition, the DS record is submitted to the parent. This is Ks' publish time (Tpub), the time that Ks is said to be Published. It is also Ks' submit time (TactDS), the time that the DS record for Ks is Submitted (ActiveDS).

Tpub(ks) >= Tgen(Ks), TactDS(Ks) == Tpub(Ks)

S(Ks) = (DNSKEY Introduced, DS Introduced)
C(ks) = Published ActiveDS

After the registration delay, the DS is published in the parent.

Event 3: The information for Ks must be published long enough to ensure that the information have reached all validating resolvers that may have the DNSKEY or DS RRset from this zone cached. At the point in time that the DNSKEY RRset including Ks has been propagated (Tkno), Ks is said to be Known. At the point in time that the DS RRset of Ks has been propagated (Tsaf), Ks is said to be SafeDS.

Tkno(Ks) >= Tpub(Ks) + DcacheK, TsafDS(Ks) >= TactDS(Ks) + DcacheP

S(Ks) = (DNSKEY Propagated, DS Propagated)
C(Ks) = Known SafeDS

Note that we could already send the request to the parent to withdraw the DS record of Kc after DcacheK. It does not matter if the DS record for Ks has not yet been propagated, since the resolver can authenticate the DNSKEY RRset with both Kc and Ks. If the validator fetches a DS RRset from the cache, it uses Kc. Otherwise, it can use Ks.

Event 4: Once we have a successor key that is said to be Known, we can withdraw the DS record for Kc. This is Kc' retire time (Tret), the time that Kc is said to be RetiredDS. If Ks is also said to be SafeDS, we no longer need to retain Kc in the zone. It is also Kc' removal time (Trem), the time that Kc is said to be Removed.

TretDS(Kc) >= Tkno(ks)
Trem(Kc) >= MAX(TsafDS(Ks), Tkno(Ks))

S(Kc) = (DNSKEY Withdrawn, DS Withdrawn)
C(Kc) = Removed RetiredDS

Event 5: From the perspective of the authoritative server, the rollover is complete. After some delay, Kc and its DS have also expired from the caches.

Tfor(Kc) >= Trem(Kc) + DcachK

TexpDS(Kc) >= TretDS(Kc) + DcacheP

S(Kc) = (DNSKEY Dead, DS Dead)
C(Kc) = Forgotten Expired

3.2.2. Double-Signature

With Double-Signature, 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.

If you want to minimize the number of interactions to the parent, this rollover method is preferred over the Double-RRset method. As a consequence, you have to wait with submitting the DS record of Ks, until it is safe to withdraw the DS record of Kc.

The timing diagram for such a rollover is:

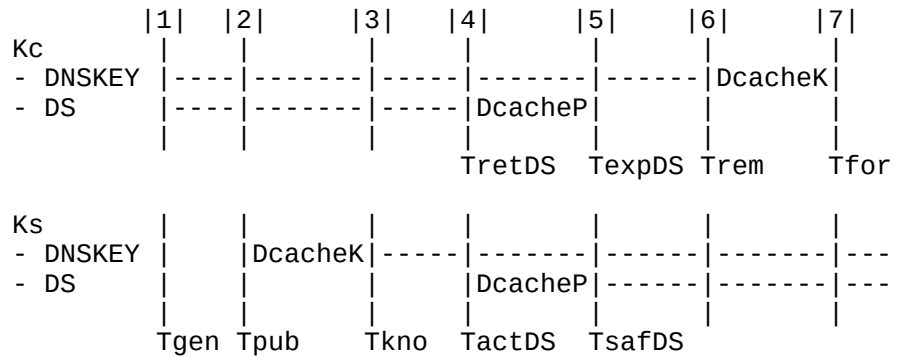


Figure: KSK Double-Signature Rollover.

Event 1: Ks is generated at time Tgen.

S(Ks) = (DNSKEY Generated, DS Generated)
C(Ks) = Generated

Event 2: Ks is introduced into the zone; it is added to the DNSKEY RRset, which is then signed by Ks and all currently actice KSKs (including Kc). This is the publication time (Tpub), the time that Ks is said to be Published.

Tpub(Ks) >= Tgen(Ks)

S(Ks) = (DNSKEY Introduced, DS Generated)

C(Ks) = Published

Event 3: Before we can submit the corresponding DS, the DNSKEY record for Ks must be published for long enough (DcacheK) to guarantee that any resolver that has a copy of the DNSKEY RRset also includes this key. In other words, that any prior cached information about the DNSKEY RRset has expired. This time is Tkno and Ks is said to be Known.

Tkno(Ks) >= Tpub(Ks) + DcacheK

S(Ks) = (DNSKEY Propagated, DS Generated)
C(Ks) = Known

Event 4: At some later time, the DS RR corresponding to Ks is submitted to the parent zone for publication. In addition, the request has been made to remove the DS RR corresponding to Kc from the parent zone. This time is Ks' submit time (TactDS), the time that Ks is considered to be Submitted. It is also Kc' retire time (TretDS), the time that Kc is considered to be RetiredDS.

TactDS(Ks) >= Tkno(Ks)
TretDS(kc) == TactDS(Kc)

S(Kc) = (DNSKEY Propagated, DS Withdrawn)
C(Ks) = Known RetiredDS
S(Ks) = (DNSKEY Propagated, DS Introduced)
C(Ks) = Known ActiveDS

After the registration delay, the DS is published in the parent.

Event 5: At some time later, all validating resolvers that have the DS RRset cached will have a a copy that includes the new DS record. This is Ks' safe time (TsafDS), the time that the new KSK is said to be SafeDS. Consequently, Kc is said to be ExpiredDS (TexpDS).

TsafDS(Ks) >= TactDS(Ks) + DcacheP
TexpDS(Kc) >= TretDS(Kc) + DcacheP

S(Kc) = (DNSKEY Propagated, DS Dead)
C(kc) = Known ExpiredDS
S(Ks) = (DNSKEY Propagated, DS Propagated)
C(Ks) = Known SafeDS

Event 6: When the new DS record has been propagated, the DNSKEY record of Kc can be removed from the zone. This is Kc' removal time (Trem), the time that Kc is said to be Removed.

Trem(Kc) >= TsafDS(Ks)

S(Kc) = (DNSKEY Withdrawn, DS Dead)
C(Kc) = Removed ExpiredDS

Event 7: From the perspective of the authoritative server, the rollover is complete. After some delay, The DNSKEY record for Kc has also expired from the caches.

Tfor(Kc) >= Trem(Kc) + DcacheK

S(Kc) = (DNSKEY Dead, DS Dead)
C(Kc) = Forgotten ExpiredDS

3.2.3. Double-DS

In this case, first 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 waiting another interval, during which the old DNSKEY RRset expires from caches, the old DS record is removed.

If you want to keep the size of the DNSKEY RRset to a minimum, this rollover method is preferred over Double-RRset. It does require the additional administrative overhead of two interactions with the parent to roll a KSK.

The timeline diagram looks like this:

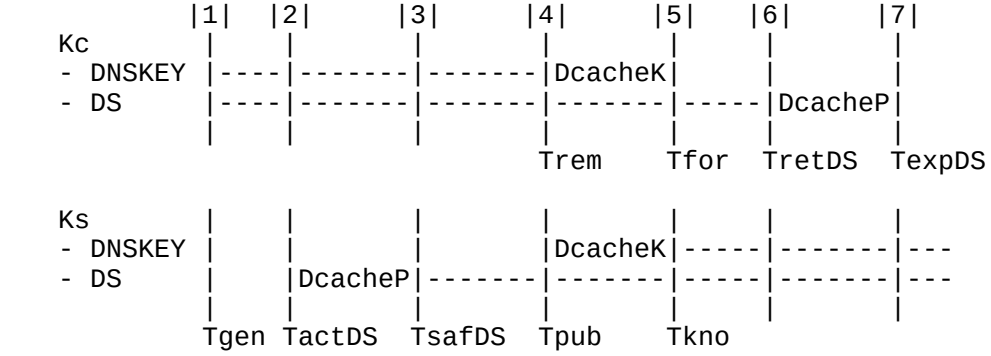


Figure: KSK Double-DS Rollover.

Event 1: Ks is generated at time Tgen.

S(Ks) = (DNSKEY Generated, DS Generated)
C(Ks) = Generated

Event 2: Before we introduce the new key Ks into the zone, we are going to submit the new DS. We can do that, because there exists a valid chain of trust for the same algorithm (Kc). This time is Ks' submit time (TactDS), the time that the DS record for Ks was submitted and is said to be ActiveDS.

TactDS(Ks) >= Tgen(Ks)

S(Ks) = (DNSKEY Generated, DS Introduced)
C(Kc) = ActiveDS

After some delay, the DS becomes available in the parent zone.

Event 3: Some time later, the new DS RRset has been propagated. This is Ks' safe time (TsafDS), the time that Ks is said to be SafeDS.

TsafDS(Ks) >= TactDS(Ks) + DcacheP

S(Ks) = (DNSKEY Generated, DS Propagated)
C(Ks) = SafeDS

Event 4: Because there are now two trust anchors a resolver can use, we can switch the KSK in the DNSKEY RRset. We stop signing with Kc and sign the DNSKEY RRset with Ks. This time is Ks' publish time (Tpub), the time that Ks is said to be Published. It is also Kc' removal time (Trem), the time that Kc is said to be Removed.

Tpub(Ks) >= TsafDS(Ks)
Trem(Kc) == Tpub(Ks)

S(Kc) = (DNSKEY Withdrawn, DS Propagated)
C(Kc) = Removed SafeDS
S(Ks) = (DNSKEY Introduced, DS Propagated)
C(Ks) = Published SafeDS

Event 5: We have to wait before Kc has been expired from the caches, before we can withdraw the DS record of Kc. When the DNSKEY RRset that includes Kc has been expired, Kc is said to be forgotten and Ks is said to be Known. This happens at Ks' known time, given by:

Tkno(Ks) >= Tpub(Ks) + DcacheK
Tfor(Kc) == Tkno(Ks)

S(Kc) = (DNSKEY Dead, DS Propagated)
C(Kc) = Forgotten SafeDS
S(Ks) = (DNSKEY Propagated, DS Propagated)
C(Ks) = Known SafeDS

Event 6: Now that we have a key Ks that is said to be Propagated and SafeDS, we are ready to withdraw the DS for Kc. We call this Kc' retire time (TretDS), the time that we don't need a secure delegation for Kc anymore.

TretDS(Kc) >= Tkno(Ks)

S(Kc) = (DNSKEY Dead, DS Withdrawn)
C(Kc) = Forgotten RetiredDS

Event 7: From the perspective of the authoritative server, the rollover is complete. After some delay, The DS record for Kc has expired from the caches. This is Texp, given by:

Texp(Kc) >= Tret(Kc) + DcacheP

S(Kc) = (DNSKEY Dead, DS Dead)
C(Kc) = Forgotten ExpiredDS

3.2.4. Interaction with Configured Trust Anchors

Zone managers may want to take into account the possibility that some validating resolvers may have their KSK configured as a trust anchor directly, as described in [RFC5011]. This influences the value of DcacheK, the time to guarantee that any resolver that has a copy of the newest DNSKEY RRset.

3.2.4.1. Adding a KSK

When the new key is introduced, the delay DcacheK between Tpub and Tkno is also subject to the condition:

DcacheK' = MAX(DcacheK, 2 * (queryInterval + x * retryTime) + c)

The right hand side of this expression is two times the Active Refresh time defined in section 2.3 in [RFC5011]. This ensures that the successor key is at least seen twice by 5011-enabled validators. The parameter x is the maximum number of retries that is taken as a safety margin, in case an Active Refresh fails. The parameter c is a constant that can be taken as an additional safety margin.

Most probably, this delays the time when a key is said to be Known.

3.2.4.2. Removing a KSK

When the current key is ready to be removed from the zone, it is instead said to be Revoked. The REVOKE bit is said and the key is published for DcacheK' time:

DcacheK' = MAX(DcacheK, (queryInterval + x * retryTime) + c)

The right hand side of this expression is the Active Refresh time defined in section 2.3 in RFC5011 [RFC5011]. This ensures that the revoked key is at least seen once by 5011-enabled validators.

After that delay, we can guarantee that every 5011-enabled resolver has seen the revoked key and it may be removed from the zone. Another DcacheK delay, the key has fully expired from all the resolver caches.

3.3. Rollovers in a Single Type Signing Scheme

In situations where you use a Single Type Signing Scheme, you can combine one of the ZSK rollover methods with one of the KSK rollover methods. However, not all combinations are possible. The KSK Double-DS rollover is only suitable for combining with the ZSK Double-RRSIG rollover, because both keep the DNSKEY RRset to a minimum size. The other ZSK rollovers require a period where both the current key and its successor are being served at the same time.

The KSK Double-RRset method is suitable with both the other ZSK rollover methods, but does not gain any advantages when combined with the ZSK Pre-Publication method. Therefore, we can leave that combination out. The KSK Double-Signature method is suitable with both the ZSK Double-Signature and the ZSK Pre-Publication method.

To conclude, we can identify four different rollover methods for the Single Type Signing Scheme.

3.3.1. Double-RRset

This is a combination of the ZSK Double-Signature rollover and the KSK Double-RRset rollover. The new KSK is added to the DNSKEY RRset, and all RRsets are 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 all zone RRsets to expire from validating resolver caches, the old DNSKEY and DS record are removed.

Double-RRset is the fastest way to replace keys in a Single Type Signing Scheme. However, it does have a lot of disadvantages of - it requires two signatures and two keys during the period of the rollover, as well as two interactions with the parent.

The timeline diagram looks like this:

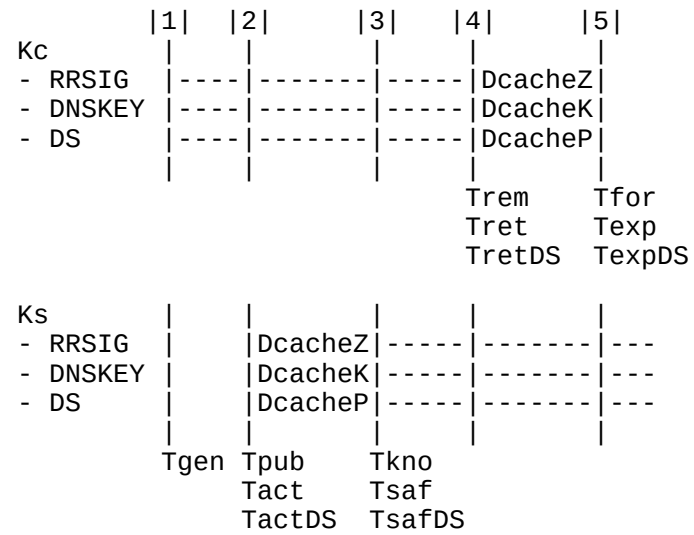


Figure: STSS Double-RRset Rollover.

The rollover method is almost the same as that of the KSK Double-RRset rollover, except we now have to take DcacheZ into account.

3.3.2. Double-Signature

This is a combination of the ZSK Double-Signature rollover and the KSK Double-Signature rollover. The new key is added to the DNSKEY RRset and all RRsets are then signed with both the old and new key. After waiting for the old RRsets 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 DNSKEY RRset, and all RRsets are signed with the new key only.

This rollover minimizes the number of interactions with the parent zone. However, for the period of the rollover all RRsets are still signed with two keys, so increasing the size of the zone and the size of the response.

The timing diagram for such a rollover is:

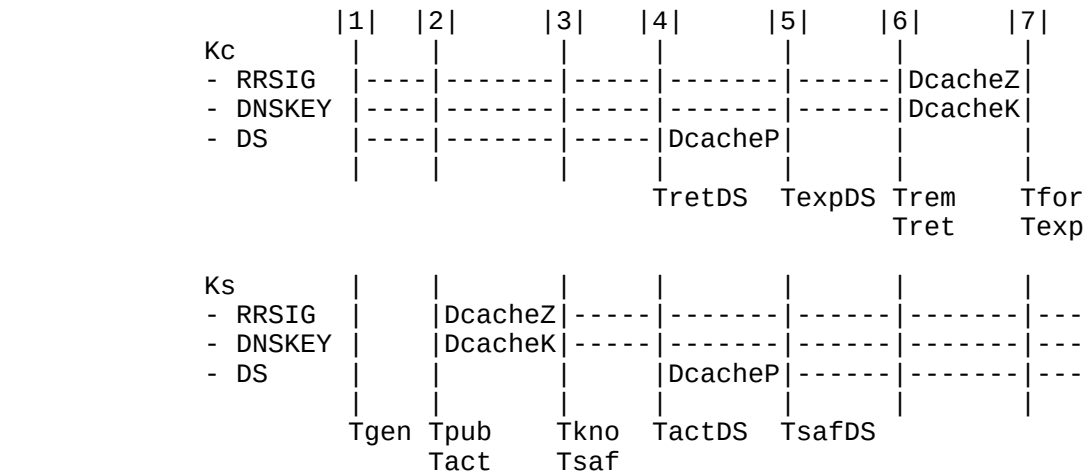


Figure: STSS Double-Signature Rollover.

The rollover diagram is almost the same as that of the KSK Double-Signature rollover, except we now have to take DcacheZ into account.

3.3.3. Pre-Publication

This is a combination of the ZSK Pre-Publication rollover and the KSK Double-Signature rollover and requires only one interaction with the parent. In addition, your non-DNSKEY RRsets require only one signature during the rollover. If speed is not an issue, this rollover method is considered to be the best practice in a Single Type Signing Scheme environment.

The new key is added to the DNSKEY RRset and the DNSKEY RRset is then signed with both the old and new key. Other RRsets will only be signed with the old key. Only after the DS has been switched, the signatures of other RRsets are replaced with that of the new key. After waiting a further interval for this change to be reflected in caches, the old key is removed from the RRset, and is signed with the new key only.

The timeline diagram looks like this:

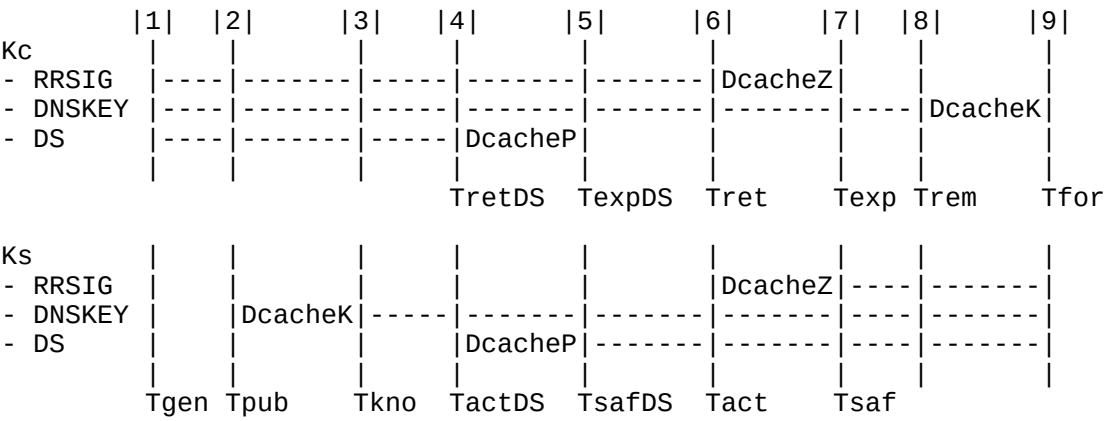


Figure: STSS Pre-Publication Rollover.

Event 1: Key Ks is generated at the generate time (Tgen).

S(Ks) = (DNSKEY Generated, RRSIG Generated, DS Generated)
C(Ks) = Generated

Event 2: The DNSKEY record of Ks is put into the zone, i.e. it is added to the DNSKEY RRset which is then re-signed with the Ks and all other current KSKs (including Kc). The time at which this occurs is Ks' publication time (Tpub), and the key is now said to be Published. Note that the key is not yet used to sign other RRsets.

Tpub(Ks) >= Tgen(Ks)

S(Ks) = (DNSKEY Introduced, RRSIG Generated, DS Generated)
C(Ks) = Published

Event 3: Before we can switch the DS, the DNSKEY record for Ks must be published for long enough (DcacheK) to guarantee that any resolver that has a copy of the DNSKEY RRset also includes this key. After this delay, the key is said to be Known and the DS record may be submitted. The time at which this event occurs is Ks' known time (Tkno), which is given by:

Tkno(Ks) >= Tpub(Ks) + DcacheK

S(Ks) = (DNSKEY Propagated, RRSIG Generated, DS Generated)
C(ks) = Known

Event 4: The time that the DS record of Ks is submitted is at Ks'

submit time (TactDS). Ks is said to be ActiveDS. At the same time, the DS record of Kc is withdrawn (TretDS) and Kc is said to be RetiredDS.

TactDS(Ks) >= Tkno(Ks) TretDS(Kc) == TactDS(Ks)

S(Kc) = (DNSKEY Propagated, RRSIG Propagated, DS Withdrawn)
C(Kc) = Known Safe RetiredDS
S(Ks) = (DNSKEY Propagated, RRSIG Generated, DS Introduced)
C(Ks) = Known ActiveDS

Some time later, the new DS RRset is published at the parent.

Event 5: Some time later, we can guarantee that all validating resolvers use the DS RRset that includes a copy of the DS record of DS. At this time, Ks' safe time (TsafDS), Ks is said to be SafeDS. But we still use Kc as ZSK.

TsafDS(Ks) >= TactDS(Ks) + DcacheP
TexpDS(Kc) >= TretDS(Kc) + DcacheP

S(Kc) = (DNSKEY Propagated, RRSIG Propagated, DS Dead)
C(kc) = Known Safe ExpiredDS
S(Ks) = (DNSKEY Propagated, RRSIG Generated, DS Propagated)
C(Ks) = Known SafeDS

Event 6: At some point in time, the decision is made to actually 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. This point in time is Ks' active time (Tact), the time that Ks is said to be Active. It is also Kc' retire time (Tret), the time that Kc is said to be Retired.

Tact(Ks) >= TsafDS(Ks)
Tret(Kc) == Tact(Ks)

S(Kc) = (DNSKEY Propagated, RRSIG Withdrawn, DS Dead))
C(Kc) = Known Retired ExpiredDS
S(Ks) = (DNSKEY Propagated, RRSIG Introduced, DS Propagated))
C(Ks) = Known Active SafeDS

Event 7: Kc needs to be retained in the zone whilst any RRSIG records created by the retired key are still published in the zone or held in resolver caches. In other words, Kc should be retained in the zone until all RRSIG records created by Ks have been propagated. This time is Ks' safe time (Tsaf), the time that Ks is considered to be Safe, and Kc' expiration time (Texp), the time that Kc is considered to be Expired.

Tsaf(Ks) >= Tact(Ks) + DcacheZ
Texp(Kc) == Tsaf(Ks)

S(Kc) = (DNSKEY Propagated, RRSIG Dead, DS Dead)
C(Kc) = Known Expired ExpiredDS
S(Ks) = (DNSKEY Propagated, RRSIG Propagated, DS Propagated)
C(Ks) = Known Safe SafeDS

Event 8: When all new signatures have been propagated, Kc can be removed from the zone and the DNSKEY RRset re-signed with the current KSK. This time is Kc' removal time (Trem), the time that Kc is considered to be Removed.

Trem(Kc) >= Tsaf(Ks)

S(Kc) = (DNSKEY Withdrawn, RRSIG Dead, DS Dead)
C(Kc) = Removed Expired ExpiredDS

Event 9: From the perspective of the authoritative server, the rollover is complete. After some delay, The DNSKEY record for Kc has expired from the caches. This is Tfor, the time that the key is said to be Forgotten.

Tfor(Kc) >= Trem(Kc) + DcacheK

S(Kc) = (DNSKEY Dead, RRSIG Dead, DS Dead)
C(Kc) = Forgotten Expired ExpiredDS

3.3.4. Double-DS

This is a combination of the ZSK Double-RRSIG rollover and the KSK Double-DS rollover. This keeps your DNSKEY RRset to a minimum size, but at the cost of double signatures in your zone and double DS at the parent.

The new signatures are added to the zone and the new DS is submitted. Once all signatures and the DS record have been propagated, the DNSKEY is switched. After waiting a further interval for this switch to be reflected in caches, the old signatures are removed and the old DS is withdrawn from the parent zone.

The timeline diagram looks like this:

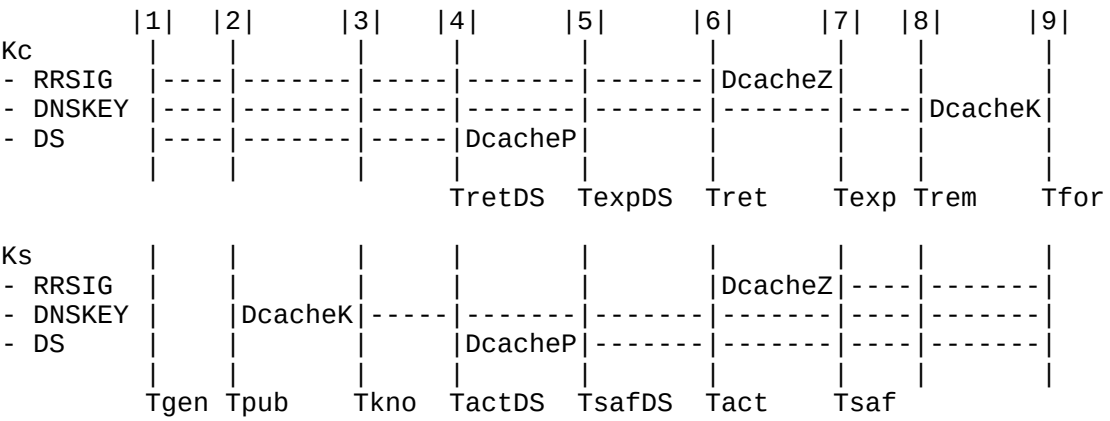


Figure: STSS Double-DS Rollover.

Event 1: Key Ks is generated at the generate time (Tgen).

S(Ks) = (DNSKEY Generated, RRSIG Generated, DS Generated)
C(Ks) = Generated

Event 2: Before we introduce the new key Ks into the zone, we are going to add the new signatures and submit the new DS. This time is Ks' active time (Tact), the time that Ks is said to be Active. It is also Ks' submit time (TactDS), the time that the DS record for Ks was submitted and is said to be ActiveDS.

Tact(Ks) >= Tgen(Ks)
TactDS(Ks) >= Tgen(Ks)

S(Ks) = (DNSKEY Generated, RRSIG Introduced, DS Introduced)
C(Kc) = Active ActiveDS

After some delay, the DS becomes available in the parent zone.

Event 3: Some time later, the new signatures and the new DS RRset have been propagated. This is Ks' safe time (Tsaf, TsafDS), the time that Ks is said to be Safe and SafeDS.

Tsaf(Ks) >= Tact(Ks) + DcacheZ
TsafDS(Ks) >= TactDS(Ks) + DcacheP

S(Ks) = (DNSKEY Generated, RRSIG Propagated, DS Propagated)
C(Ks) = Safe SafeDS

Event 4: Because there are now two trust anchors a resolver can use, we can switch the KSK in the DNSKEY RRset. This time is Ks' publish time (Tpub), the time that Ks is said to be Published. It is also Kc' removal time (Trem), the time that Kc is removed from the zone.

```
Tpub(Ks) >= MAX(TsafDS(Ks), Tsaf(Ks))
Trem(Kc) == Tpub(Ks)

S(Kc) = (DNSKEY Withdrawn, RRSIG Propagated, DS Propagated)
C(Kc) = Removed Safe SafeDS
S(Ks) = (DNSKEY Introduced, RRSIG Propagated, DS Propagated)
C(Ks) = Published Safe SafeDS
```

Event 5: We have to wait before the signatures of Kc and its corresponding DS record have been expired from the caches, before we can withdraw the DNSKEY record of Kc. When the DNSKEY RRset that includes Kc has been expired, Ks is said to be Known and Kc is said to be Removed. This happens at Ks' known time, given by:

```
Tkno(Ks) >= Tpub(Ks) + DcacheK, Trem(Kc) == Tkno(Ks)

S(Kc) = (DNSKEY Dead, RRSIG Propagated, DS Propagated)
C(Kc) = Forgotten Safe SafeDS
S(Ks) = (DNSKEY Propagated, RRSIG Propagated, DS Propagated)
C(Ks) = Known Safe SafeDS
```

Event 6: Now that we have a key Ks that is said to be Propagated and SafeDS, we are ready to withdraw the signatures and DS for Kc. We call this Kc' retire time (Tret, TretDS), the time Kc is said to be Retired and RetiredDS.

```
Tret(Kc) >= Tkno(Ks)
TretDS(Kc) >= Tkno(Ks)

S(Kc) = (DNSKEY Dead, RRSIG Withdrawn, DS Withdrawn)
C(Kc) = Forgotten Retired RetiredDS
```

Event 7: From the perspective of the authoritative server, the rollover is complete. After some delay, The signatures of Kc and its corresponding DS record have expired from the caches.

```
Texp(Kc) >= Tret(Kc) + DcacheZ
TexpDS(Kc) >= TretDS(Kc) + DcacheP

S(Kc) = (DNSKEY Dead, RRSIG Dead, DS Dead)
C(Kc) = Forgotten Expired ExpiredDS
```

3.4. Stand-by 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 ready to sign the zone, having at least one additional key (a stand-by key) in this state at all times will minimise delay.

In the case of a ZSK, a stand-by key only makes sense with the Pre-Publication method, since with the Double-Signature and Double-RRSIG methods, the stand-by key would be used for signing. The goal is to make the stand-by key Known. This goal is reached at Tkno, step 3 in the Pre-Publication method timeline diagram.

A successor key must always be published soon enough so that the key lifetime of the predecessor key does not exceed. That means that the successor ZSK Ks must at latest be published DcacheK delay before the lifetime of the predecessor ZSK kc has reached:

Tpub(Ks) <= Tact(Kc) + Lzsk - DcacheK

Here, Lzsk is the lifetime of ZSKs according to policy.

In the case of a KSK, a stand-by key only makes sense with the Double-DS method, since in the other cases, the key would be needed to sign the DNSKEY RRset. The goal is to get the stand-by key in the SafeDS condition. This goal is reached at TsafDS, step 3 in the Double-DS method timeline diagram.

The DS record for the successor KSK Ks should be propagated to the caches before the key lifetime of the predecessor KSK Kc exceeds:

TactDS(Ks) <= Tact(Kc) + Lksk - DcacheP

Here, Lksk is the lifetime of KSKs according to policy.

Because a stand-by KSK only makes sense with the Double-DS method, stand-by keys in a STSS is not applicable. This is because the Double-DS method is not easy integratable with one of the ZSK rollover methods.

4. Policy rollover

Besides your scheduled and unscheduled key rollovers, changes in policy may occur. The initial transition is enabling DNSSEC. The counterpart, disabling DNSSEC, is also possible. Two other policy

changes we have encountered are are algorithm rollover and changing signing schemes.

4.1. Enabling DNSSEC

When a zone makes the transition from going insecure to secure, the initial set of keys safely need to be introduced into the zone. The goals of this event is to make a ZSK (Kz) and a KSK (Kk) both Known and Safe.

One must take into account that resolver caches may hold unsigned RRsets. Therefore, validating resolvers should not know about the initial DNSKEY RRset before all unsigned RRsets have been expired from the caches. This means that the zone must be fully signed, before the DS associated with the initial KSK is published. Only if you are afraid that a key scraper fetches your DNSKEY RRset too soon, you should wait with publishing your DNSKEY RRset until enough time has elapsed for all unsigned RRsets to expire from all resolver caches. The ZSK and KSK can be the same key, for example in a Single Type Signing Scheme.

The timeline diagram is shown below:

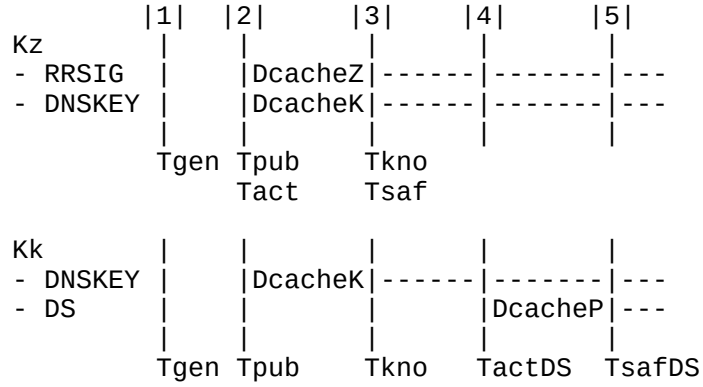


Figure: Enabling DNSSEC.

Event 1: Kk and Kz are generated. We call this Tgen, the time that the keys were Generated (note that Tgen for Kk could be different that Tgen for Kz).

S(Kk) = (DNSKEY Generated, DS Generated)
C(Kk) = Generated
S(Kz) = (DNSKEY Generated, RRSIG Generated)
C(Kk) = Generated

Event 2: The keys are put into the zone and are immediately used for signing. Because there exists no pointer to the fact that our zone is DNSSEC enabled, the DNSKEY and RRSIG records may be introduced at the same time. This is the publish time (Tpub), the time that the keys are Published. It is also Kz' active time (Tact), the time that Kz is said to be Active.

```
Tpub(Kk) >= Tgen(Kk)
Tpub(Kz) >= Tgen(Kz)
Tact(Kz) == Tpub(Kz)

S(Kk) = (DNSKEY Introduced, DS Generated)
C(Kk) = Published
S(Kz) = (DNSKEY Introduced, RRSIG Introduced)
C(Kz) = Published Active
```

Event 3: Before we can submit the DS record, Kz must be considered Known and Safe. Once that has happened, we are done for the ZSK. This time is Kz' known time (Tkno).

```
Tkno(Kk) >= Tpub(Kk) + DcacheP
Tkno(Kk) == Tkno(Kz)
Tsaf(Kz) >= Tact(Kz) + DcacheZ

S(Kk) = (DNSKEY Propagated, DS Generated)
C(Kk) = Known
S(Kz) = (DNSKEY Propagated, RRSIG Propagated)
C(Kz) = Known Safe
```

Because this is the first DNSKEY for this zone, the Dttl for the DNSKEY RRset is Ingc, the negative cache interval from the zone's SOA record, calculated according to RFC2308 [RFC2308] as the minimum of the TTL of the SOA record itself and the MINIMUM field in the record's parameters:

```
Ingc = min(TTL(SOA), MINIMUM)
```

Event 4: Once we are sure of the fact that the DNSKEY RRset and all RRSIG records have reached the caches, we may submit the DS to the parent. We call this TactDS, the time that the DS has been submitted to the parent.

```
TactDS(Kk) >= Tkno(Kk)

S(Kk) = (DNSKEY Propagated, DS Introduced)
C(Kk) = Known ActiveDS
```

Event 5: Some time later, the DS has been published in the parent

zone. Some more time later, all resolvers that have a copy of the DS RRset have one that includes the DS record of Kk.

TsafDS(Kk) >= TactDS(Kk) + DcacheP

S(Kk) = (DNSKEY Propagated, DS Propagated)
C(Kk) = Known SafeDS

Because this is the first DS for this zone, the Dttl for the DS RRset is Ingc, for the same reason as in step 3 for the DNSKEY RRset.

4.2. Disabling DNSSEC

When a zone decides for whatever reason to go back to the Insecure status, the set of keys safely need to be removed from the zone. We assume that there is a KSK (Kk) and a ZSK (Kz) that are Known and Safe. The goals of this event is to make Kk and Kz both Forgotten and Expired.

The timeline diagram is shown below:

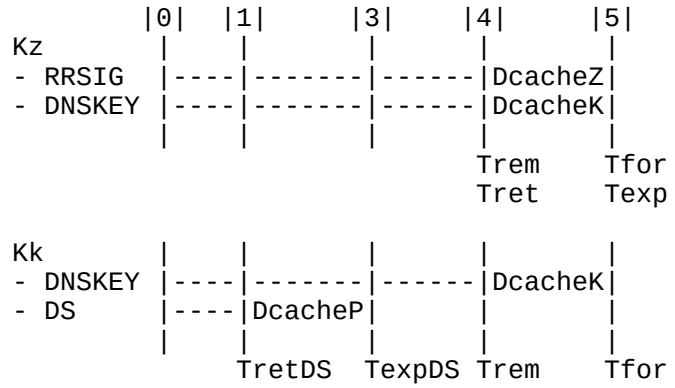


Figure: Disabling DNSSEC.

Event 1: The DS record of Kk needs to be withdrawn. This time is Kk' retire time (TretDS), the time that Kk is said to be RetiredDS.

S(Kk) = (DNSKEY Propagated, DS Withdrawn)
C(Kk) = Known RetiredDS

Event 2: We have to wait until the DS record of Kk has expired from all resolver caches. This time is Kk' expire time (TexpDS), the time that Kk is said to be ExpiredDS.

TexpDS(Kk) >= TretDS(Kk) + DcacheP

S(Kk) = (DNSKEY Propagated, DS Dead)
C(Kk) = Known ExpiredDS

Event 3: Now that we can guarantee that no secure chain of trust to Kk exist anymore, we can retire the ZSK and withdraw both keys. This time is Trem, the time that the keys are removed from the zone.

Trem(Kk) >= TexpDS(Kk)
Trem(Kz) == Trem(Kk)
Tret(Kz) == Trem(kz)

S(Kk) = (DNSKEY Withdrawn, DS Dead)
C(Kk) = Removed ExpiredDS
S(Kz) = (DNSKEY Withdrawn, RRSIG Withdrawn)
C(Kz) = Removed Retired

Event 4: After some delay, all information about the keys have expired from the caches.

Tfor(Kk) >= Trem(Kk) + DcacheK
Tfor(Kz) == Tfor(Kk)
Texp(Kz) >= Tret(Kz) + DcacheZ

S(Kk) = (DNSKEY Dead, DS Dead)
C(Kk) = Forgotten ExpiredDS
S(Kz) = (DNSKEY Dead, RRSIG Dead)
C(Kz) = Forgotten Expired

4.3. Algorithm Rollover

When changing algorithms, you can either add, remove or replace an algorithm. Adding and removing an algorithm follow the same timings as enabling and disabling DNSSEC. Replacing an algorithm can be done with a STSS Double-Signature rollover or a KSK and ZSK Double-Signature Rollover at the same time. [MM: This needs more text, but I am awaiting the discussion about algorithm rollover and how to interpret section 2.2 of RFC 4035]

4.4. KSK-ZSK Split or Single Type Signing Scheme

When changing signing schemes, you should follow the timelines of the most restricting signing scheme. The STSS signing scheme makes some rollover combinations unsuitable, thus it can be considered the most restricted signing scheme. In the case of moving to a KSK-ZSK Split, Ks is used as the successor key in the STSS rollover methods, and it now reflects both the successor ZSK and KSK. In the case of moving

away from a KSK-ZSK Split, Kc is used as the predecessor key in the STSS rollover methods, and it now reflects both the predecessor ZSK and KSK. [MM: This could perhaps also use more explanation.]

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

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

7. Acknowledgements

Special acknowledgments and gratitude go out to Stephen Morris, Johan Ihren and John Dickinson, the authors of the key-timing draft [key-timing]. Significant parts of the text is taken from that document. Especially Section 3.1 and Section 3.2 are largely copied and adjusted to the new introduced terminology from this document.

I also want to acknowledge Yuri Schaeffer, who brought to my attention the idea of key goals (Section 1.1.1) and whose discussions helped to shape this document.

8. Changes with key-timing draft

This document builds further on draft-ietf-dnsop-dnssec-key-timing [key-timing]. The most important changes with respect to that document are:

- Introduced the concept of Rollover Considerations (Speed vs Size vs Interactions), that causes the existence of different key rollover scenarios.

- Introduced the concept of Key Goals.

- Key States are unraveled to represent the status of each piece of information seperately. Provides more flexibility. Used for combining rollover methods in a Single Type Singing Scheme.

- What were Key States in the key-timing draft, are now called Key Conditions. A key can have more than one condition.

- Four new Key Conditions are introduced: Known, Safe, Forgotten and Expired, to represent whether information about the key exist in resolver caches. The key conditions Ready and Dead are deprecated.

- Timelines for STSS Rollovers.

- Timelines for enabling and disabling DNSSEC.

- Text about policy rollover, such as algorithm rollover and changing signing schemes.

9. References

9.1. Informative References

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

9.2. 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.

[dnssec-bis] Weiler, S. and D. Blacka, "Clarifications and Implementation Notes for DNSSECbis", November 2010.

[key-timing] Morris, S., Ihren, J., and J. Dickinson, "DNSSEC Key Timing Considerations", July 2010.

Author's Address

Matthijs Mekking
NLnet Labs
Science Park 140
Amsterdam 1098 XG
The Netherlands

EMail: matthijs@nlnetlabs.nl

Individual Submission
Internet-Draft
Intended status: BCP
Expires: September 8, 2011

G. Michaelson
G. Huston
APNIC

March 7, 2011

AS112 Nameserver Delegations for IPv6
draft-michaelson-as112-ipv6-00

Abstract

To reduce longterm traffic to the DNS root servers and the IP6.ARPA authoritative servers, the IAB is requested to instruct the IANA to delegate a set of sub-domains of IP6.ARPA to the AS112 Project [ID.ietf-dnsop-as112-ops]. These domains represent IPv6 address prefixes that are not conventionally populated in the global reverse-DNS, including IPv6 prefixes that are not globally scoped and certain prefixes used in an anycast context.

The reverse DNS query load associated with these IPv6 address prefixes appear to have unacceptable scaling consequences as IPv6 uptake increases. By delegating these sub-domains to the AS112 project, the DNS query load can be passed to a distributed sink, reducing the query load on the root servers and the IP6.ARPA authoritative servers.

Table of Contents

1. Reverse DNS Delegation and Local-Use Addresses

The IPv6 Addressing Architecture [RFC4291] includes certain address prefixes that are not intended to be uniquely used in the global network as globally-scoped unicast addresses. Such addresses include locally-scoped addresses, certain anycast addresses, and loopback addresses.

While such addresses are not intended to be used in the same context as globally-scoped unicast addresses, their use in various local and global contexts is seen to trigger Domain Name System (DNS) [RFC1034] queries (of the form of "reverse lookups") corresponding to these addresses. Since the addresses concerned generally have local rather than global significance, it is good practice for site administrators to ensure that such queries are answered locally [I-D.ietf-dnsop-default-local-zones]. However, it is not uncommon for such queries to follow the normal delegation path in the public DNS instead of being answered within the site. It is not possible for public DNS servers to give useful answers to such queries, and the response to such reverse lookup queries from the global DNS is the "Name Error" RCODE described in [RFC1035], commonly termed "NXDOMAIN".

When the reverse-DNS infrastructure receives a request for un-delegated sub-domains, the point of delegation of the last matched label along the name path to the root receives the query. In the case of the IPv6 reverse delegation structure, this implies that the IP6.ARPA authoritative servers will receive the query load. Because the sub-domain is not delegated, the server is obliged to answer with an NXDOMAIN response. Since negative caching is not widely deployed, a large number of these DNS queries are repeated, further increasing the DNS query load imposed on the DNS root servers and the IP6.ARPA authoritative servers.

This query load appears to have unacceptable scaling consequences as IPv6 uptake increases. By delegating these sub-domains to the AS112 project [ID.ietf-dnsop-as112-ops], the DNS query load can be passed off to a distributed dedicated server set, reducing the load on the DNS root and the IP6.ARPA authoritative servers.

2. IANA Considerations

As per the provisions of [RFC3152], this document recommends the IAB to direct IANA to delegate the following IP6.ARPA reverse DNS zones to the AS112 project [ID.ietf-dnsop-as112-ops]:

```
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.ip6.arpa (Unspecified)
                              f.f.ip6.arpa (Multicast)
                            8.e.f.ip6.arpa (Link-Local Scope)
                            9.e.f.ip6.arpa (Link-Local Scope)
                            a.e.f.ip6.arpa (Link-Local Scope)
                            b.e.f.ip6.arpa (Link-Local Scope)
                            c.e.f.ip6.arpa (Link-Local Scope)
                            d.e.f.ip6.arpa (Link-Local Scope)
                            e.e.f.ip6.arpa (Link-Local Scope)
                            f.e.f.ip6.arpa (Link-Local Scope)
                          0.0.c.f.ip6.arpa (Unique Locally Assigned)
                          0.0.d.f.ip6.arpa (Unique Locally Assigned)
                  0.0.0.0.1.0.0.2.ip6.arpa (Teredo)
```

AS112 project servers should add these zones to their configuration, and terminate queries efficiently inside their service infrastructure.

This delegation instruction is subject to further direction in the future from the IAB to IANA, as per the provisions of [RFC3152].

3. Security Considerations

The Security Considerations described in [ID.ietf-dnsop-as112-ops] also apply to local-use IPv6 addresses, and should be considered in the context of the use of these addresses.

DNS queries may well identify the location of deployment of IPv6 enabled equipment in private contexts, particularly when the reverse queries relate to local-use IPv6 addresses. While operators of the DNS reverse servers should respect the privacy of data relating to individual queries made to these reverse address servers, the unintentional leakage of information beyond its intended scope of use and circulation represents a potential threat to the security of a local private network. This direction to delegate these local-use IPv6 reverse address sub-domains does not substantially change the security risks of information leakage from private environments.

4. Acknowledgments

The authors acknowledge the work of Joe Abley and William Maton and the DNSOPS Working Group in preparing the AS112 framework document for delegation of the private use address blocks in IPv4, and have used parts of their AS112 document as a template for these AS112 delegation instructions in IPv6.

5. References

5.1. Normative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

5.2. Informative References

[I-D.ietf-dnsop-default-local-zones] Andrews, M., "Locally-served DNS Zones", Internet Draft draft-ietf-dnsop-default-local-zones, September 2010.

[ID.ietf-dnsop-as112-ops] Abley, J. and W. Maton, "Locally-served DNS Zones", Internet Draft draft-ietf-dnsop-default-local-zones, November 2010.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC3152] Bush, R., "Delegation of IP6.ARPA", BCP 49, RFC 3152, August 2001.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

Authors' Addresses

George Michaelson
APNIC

Email: ggm@apnic.net
URI: http://www.apnic.net

Geoff Huston
APNIC

Email: gih@apnic.net
URI: http://www.apnic.net