Internet Engineering Task Force (IETF)
Request for Comments: 6781
Obsoletes: 4641
Category: Informational
ISSN: 2070-1721

O. Kolkman
W. Mekking
NLnet Labs
R. Gieben
SIDN Labs

December 2012

DNSSEC Operational Practices, Version 2

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.

Status of This Memo

Copyright Notice

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 [RFC1034] and RFC 1035 [RFC1035]) and want to deploy DNSSEC (RFC 4033 [RFC4033], RFC 4034 [RFC4034], RFC 4035 [RFC4035], and RFC 5155 [RFC5155]). 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 relationship 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. Although the DNS Root has been signed since July 15, 2010 and now more than 80 secure delegations are provisioned in the root, at the time of this writing there still exists relatively little experience with DNSSEC in production environments below the Top-Level Domain (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 the 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. Sections 4.1 and 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 key publication. Section 5 covers the considerations regarding selecting and using the NSEC or NSEC3 [RFC5155] Resource Record.

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

Since we describe operational suggestions and there are no protocol specifications, the RFC 2119 [RFC2119] language does not apply to this document, though we do use quotes from other documents that do include the RFC 2119 language.

This document obsoletes RFC 4641 [RFC4641].

1.1. The Use of the Term ’key’

It is assumed that the reader is familiar with the concept of asymmetric cryptography, or public-key cryptography, on which DNSSEC is based (see the definition of ’asymmetric cryptography’ in RFC 4949 [RFC4949]). 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 (DNSKEY RR) and that it is the public part that is used in signature verification.

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 (absolute) time specified in the signature inception field of the RRSIG RR and ends at the (absolute) time specified in the expiration field of the RRSIG RR. The document sometimes also uses the term ’validity period’, which means the same.

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 earliest 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, that are used by validators or resolvers. Note that the minimum TTL is not the same as the MINIMUM field in the SOA RR. See RFC 2308 [RFC2308] 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 RFC 4033 [RFC4033] 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 name servers and clients may be fetching data from caching non-authoritative servers. In this light, note that the time until the data is available on the slave can be negligible when using NOTIFY [RFC1996] and Incremental Zone Transfer (IXFR) [RFC1995]. It increases when Authoritative (full) Zone Transfers (AXFRs) are used in combination with NOTIFY. It increases even more if you rely on the full zone transfers being based only on 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 name server, 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 tradeoff 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. Key Generation and Storage

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

- Does one differentiate between Zone Signing Keys 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 [RFC4033]?

- 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 Keys 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 cases where the differentiation between the KSK and ZSK is not made, i.e., where keys have the role of both 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 DNSKEY RRset is signed with the KSK, all the keys in the RRset can be used as ZSKs. If there has been an event that increases the risk that a ZSK is compromised, it can be simply replaced with a ZSK rollover. The new RRset is then re-signed with the KSK.

Changing a key that is a Secure Entry Point (SEP) [RFC4034] for a zone can be relatively expensive, as it involves interaction with third parties: When a key is only pointed to by a Delegation Signer (DS) [RFC4034] record in the parent zone, one needs to complete the interaction with the parent 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, it is a KSK; otherwise, it is a ZSK.

There is also a risk that keys can be compromised through theft or loss. For keys that are installed on file-systems of name servers 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 limitations 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 involved costs. For example, a KSK can be stored off-line or with more limitations 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 file-system accessible by operators for daily routine use, may provide 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 operational flexibility. In those cases, a KSK-ZSK split is not more beneficial than the Single-Type Signing Scheme.

It is worth mentioning that there’s not much point in obsessively protecting the key if you don’t protect the zone files, which also live on the file-systems.

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 be somewhere on the order of 10 to 100, the associated key sizes will vary only by about a byte in size for symmetric keys. When translated to asymmetric keys, the size difference is still too insignificant 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 child-parent provisioning chain -- in particular, the timely appearance of a new DS record in the parent zone in emergency situations -- is predictable.

If the above arguments hold, 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.

3.2. Practical Consequences of KSK and ZSK Separation

A key that acts only as a Zone Signing Key is used to sign all the data except 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 a relatively short key effectivity period.

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. These can include the administrators 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 (e.g., RFC 5011 [RFC5011]) 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 three schools of thought on rolling a KSK that is not a trust anchor:

1. It should be done frequently and regularly (possibly every few months), so that a key rollover remains an operational routine.

2. It should be done frequently but irregularly. "Frequently" means every few months, again based on the argument that a rollover is a practiced and common operational routine; "irregular" means with a large jitter, so that third parties do not start to rely on the key and will not be tempted to configure it as a trust anchor.

3. It should only be done when it is known or strongly suspected that the key can be or has been compromised, or in conjunction with operator change policies and procedures, like when a new algorithm or key storage is required.

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 administrator 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 administrator, there may be stability costs for the validator administrator that (wrongfully) configured the trust anchor when the zone administrator rolls 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 administrator publishes a DNSSEC signing policy and/or a DNSSEC practice statement [DNSSEC-DPS], that policy or statement should be explicit regarding whether or not the existence of trust anchors will be taken into account. There may be cases where local policies enforce the configuration of trust anchors on zones that 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 administrators 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 are smaller. In fact, the costs for those users can be minimized by automating the rollover with RFC 5011 [RFC5011] and by rolling the key regularly (and advertising such) so that the operators of validating resolvers will put the appropriate mechanism in place to deal with these stability costs: In other words, budget for these costs instead of incurring them unexpectedly.

It is therefore preferable to roll KSKs that are expected to be used as trust anchors on a regular basis if and only if those rollovers can be tracked using standardized (e.g., RFC 5011 [RFC5011]) mechanisms.

3.2.3. The Use of the SEP Flag

The so-called SEP [RFC4035] 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, i.e., 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 validation, it is used in practice for operational purposes such as for the rollover mechanism described in RFC 5011 [RFC5011]. 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 suggested that the SEP flag be 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 a KSK and ZSK is not made (i.e., for a Single-Type Signing Scheme), it is suggested that the SEP flag be set on all keys.

Note: Some 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 on the order of two 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 opts 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 an 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 online 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 discussed in Sections 4.1 and 4.4. Key replacement endures for a couple of Maximum Zone TTLs, depending on the rollover scenario. Therefore, a multiple of Maximum Zone TTL durations is a reasonable lower limit on the key effectivity period. Forcing a shorter key effectivity period will result in an unnecessary and inconveniently large DNSKEY RRset published in the zone.

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. Thus, in cases where the ZSK cannot be afforded the same level of protection as the KSK (such as when zone keys are kept online), and where the risk of unauthorized disclosure of the ZSK’s private key is not negligible (e.g., when HSMs are not in use), the ZSK’s effectivity period should be kept shorter than the KSK’s effectivity period.

In fact, if the risk of loss, theft, or other compromise is the same for a ZSK and a KSK, 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 in which the use of a Single-Type Signing Scheme with a long key effectivity period is a good choice, for example, where the costs and risks of compromise, and the costs and risks involved with having to perform an emergency roll, are low.

3.4. Cryptographic Considerations

3.4.1. Signature Algorithm

At the time of this writing, there are three types of signature algorithms that can be used in DNSSEC: RSA, Digital Signature Algorithm (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 note that, in the context of DNSSEC, DSA is limited to a maximum of 1024-bit keys.

We suggest the use of RSA/SHA-256 as the preferred signature algorithm 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 be targeted for an effective attack.

At the time of publication, it is known that the SHA-1 hash has cryptanalysis issues, and work is in progress to address them. The use of public-key algorithms based on hashes stronger than SHA-1 (e.g., SHA-256) is recommended, if these algorithms are available in implementations (see RFC 5702 [RFC5702] and RFC 4509 [RFC4509]).

Also, at the time of publication, digital signature algorithms based on Elliptic Curve (EC) Cryptography with DNSSEC (GOST [RFC5933], Elliptic Curve Digital Signature Algorithm (ECDSA) [RFC6605]) are being standardized and implemented. The use of EC has benefits in terms of size. On the other hand, one has to balance that against the amount of validating resolver implementations that will not recognize EC signatures and thus treat the zone as insecure. Beyond the observation of this tradeoff, we will not discuss this further.

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), it may be advisable 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 (RFC 3766 [RFC3766]). Signing and verifying with a 2048-bit key takes 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) are about eight times slower.

Another way to decide on the size of a 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 Transport Layer Security (TLS) [RFC5246] 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 built 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 preferred 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 to the master.

When relying on dynamic update [RFC3007] 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 on the level of security of the host. Although not mandatory, one could administer a zone using a "hidden master" scheme to minimize the risk. In this arrangement, the master name server that processes the updates is unavailable to general hosts on the Internet; it is not listed in the NS RRset. The name servers 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 the master copy against unauthorized access in order to prevent modification of DNS data before it is 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 user).
- The costs of rolling the (compromised) keys. (The cost of rolling a ZSK is lowest, and the cost 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 [RFC3007], 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 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 [RFC4086] and NIST SP 800-90A [NIST-SP-800-90A]. 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 preferred 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?

An earlier version of this document (RFC 4641 [RFC4641]) made a differentiation between key lengths for KSKs used for zones that are high in the DNS hierarchy and those for KSKs used lower down in the hierarchy.

This distinction is now considered irrelevant. 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.

4. Signature Generation, Key Rollover, and Related Policies

4.1. Key Rollovers

Regardless of whether a zone uses periodic key rollovers or only rolls keys in case of an irregular event, 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 the fact 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.

The typographic conventions used in the diagrams below are explained in Appendix B.

4.1.1. Zone Signing Key Rollovers

If the choice for splitting ZSKs and KSKs has been made, then those two types of keys can be rolled separately, and ZSKs can be rolled without taking into account DS records from the parent or the configuration of such a key as the 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 scheme, described in Section 4.1.1.1, uses key pre-publication; the other uses double signatures, as described in Section 4.1.1.2. The pros and cons 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 from DNSKEY_Z_10 to DNSKEY_Z_11 involves four stages as follows:

```
------------------------------------------------------------
 initial            new DNSKEY          new RRSIGs
------------------------------------------------------------
 SOA_0              SOA_1               SOA_2
 RRSIG_Z_10(SOA)    RRSIG_Z_10(SOA)     RRSIG_Z_11(SOA)

 DNSKEY_K_1         DNSKEY_K_1          DNSKEY_K_1
 DNSKEY_Z_10        DNSKEY_Z_10         DNSKEY_Z_10
                    DNSKEY_Z_11         DNSKEY_Z_11
 RRSIG_K_1(DNSKEY)  RRSIG_K_1(DNSKEY)   RRSIG_K_1(DNSKEY)
------------------------------------------------------------

------------------------------------------------------------
 DNSKEY removal
------------------------------------------------------------
 SOA_3
 RRSIG_Z_11(SOA)

 DNSKEY_K_1
 DNSKEY_Z_11

 RRSIG_K_1(DNSKEY)
------------------------------------------------------------
```

Figure 1: Pre-Publish Key Rollover

initial: Initial version of the zone: DNSKEY_K_1 is the Key Signing Key. DNSKEY_Z_10 is used to sign all the data of the zone, i.e., it is 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 its 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 the TTL value of the key set.

new RRSIGs: At the "new RRSIGs" stage, 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 the zone in the "new DNSKEY" step can still be verified with key sets fetched from this version 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 DNSKEY_K_1.

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)
       -----------------------------------------------------------------

       -----------------------------------------------------------------
       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)
       -----------------------------------------------------------------
```

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 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 propagation delay plus the Maximum Zone TTL of previous versions of the zone.

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)

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, i.e., it is the Zone Signing Key.

new DNSKEY: At the "new DNSKEY" stage, 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 (i.e., the version of the zone data containing SOA_0) of the zone has been replaced in all secondary servers and then has expired from remote caches. This will take at least the propagation delay plus the Maximum Zone TTL of version 0 of the zone.

DNSKEY removal: DNSKEY_Z_10 is removed from the zone, as are all signatures created with it. 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 vice versa. The duration of the "new DNSKEY" phase and the period between rollovers should be at least the propagation delay to secondary servers plus the Maximum Zone TTL of the previous version of the zone.

Note that in this example we assumed for simplicity that the zone was not modified during the rollover. In fact, new data can be introduced at any time during this period, 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 stages. Also, the Pre-Publish scheme involves more parental work when used for KSK rollovers, as explained in Section 4.1.2.

Double-Signature ZSK rollover: The drawback of this approach 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 stages.

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.

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

```
   ---------------------------------------------------------------------
    initial            new DNSKEY        DS change    DNSKEY removal
   ---------------------------------------------------------------------
   Parent:
    SOA_0 -----------------------------> SOA_1 ------------------------>
    RRSIG_par(SOA) --------------------> RRSIG_par(SOA) --------------->
    DS_K_1 ----------------------------> DS_K_2 ----------------------->
    RRSIG_par(DS) ---------------------> RRSIG_par(DS) ---------------->

   Child:
    SOA_0              SOA_1 -----------------------> SOA_2
    RRSIG_Z_10(SOA)    RRSIG_Z_10(SOA) -------------> RRSIG_Z_10(SOA)

    DNSKEY_K_1         DNSKEY_K_1 ------------------>
                       DNSKEY_K_2 ------------------> DNSKEY_K_2
    DNSKEY_Z_10        DNSKEY_Z_10 -----------------> DNSKEY_Z_10
    RRSIG_K_1(DNSKEY)  RRSIG_K_1 (DNSKEY) ---------->
                       RRSIG_K_2 (DNSKEY) ----------> RRSIG_K_2(DNSKEY)
   ---------------------------------------------------------------------
```

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 relationship, 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. This is known as a KSK Double-DS rollover and is shown in Figure 5. This method has some drawbacks for KSKs. We first describe the rollover scheme and then indicate these drawbacks.

```
--------------------------------------------------------------------
  initial         new DS         new DNSKEY       DS removal
--------------------------------------------------------------------
Parent:
  SOA_0           SOA_1 ------------------------> SOA_2
  RRSIG_par(SOA)  RRSIG_par(SOA) ---------------> RRSIG_par(SOA)
  DS_K_1          DS_K_1 ----------------------->
                  DS_K_2 ----------------------> DS_K_2
  RRSIG_par(DS)   RRSIG_par(DS) ----------------> RRSIG_par(DS)

Child:
  SOA_0 -----------------------> SOA_1 ---------------------------->
  RRSIG_Z_10(SOA) -------------> RRSIG_Z_10(SOA) ------------------>

  DNSKEY_K_1 ------------------> DNSKEY_K_2 ----------------------->
  DNSKEY_Z_10 -----------------> DNSKEY_Z_10 ---------------------->
  RRSIG_K_1 (DNSKEY) ----------> RRSIG_K_2 (DNSKEY) --------------->
--------------------------------------------------------------------
```

Figure 5: Stages of Deployment for a Double-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. If the old key has expired from caches, at the "DS removal" phase the parent can be notified 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.

4.1.2.1. Special Considerations for RFC 5011 KSK Rollover

The scenario sketched above assumes that the KSK is not in use as a trust anchor but that validating name servers exclusively depend on the parental DS record to establish the zone’s security. If it is known that validating name servers have configured trust anchors, then that needs to be taken into account. Here, we assume that zone administrators will deploy RFC 5011 [RFC5011] style rollovers.

RFC 5011 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. Single-Type Signing Scheme Key Rollover

The rollover of a key 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 describe the most straightforward rollover scheme first.

```
   -------------------------------------------------------------------
     initial           new DNSKEY      DS change     DNSKEY removal
   -------------------------------------------------------------------
   Parent:
     SOA_0 --------------------------> SOA_1 ---------------------->
     RRSIG_par(SOA) -----------------> RRSIG_par(SOA) ------------->
     DS_S_1 -------------------------> DS_S_2 --------------------->
     RRSIG_par(DS_S_1) --------------> RRSIG_par(DS_S_2) ---------->

   Child:
     SOA_0             SOA_1 ----------------------> SOA_2
     RRSIG_S_1(SOA)    RRSIG_S_1(SOA) ------------->
                       RRSIG_S_2(SOA) -------------> RRSIG_S_2(SOA)
     DNSKEY_S_1        DNSKEY_S_1 ----------------->
                       DNSKEY_S_2 -----------------> DNSKEY_S_2
     RRSIG_S_1(DNSKEY) RRSIG_S_1(DNSKEY) ---------->
                       RRSIG_S_2(DNSKEY) ----------> RRSIG_S_2(DNSKEY)
   -------------------------------------------------------------------
```

Figure 6: Stages of the Straightforward Rollover
in a Single-Type Signing Scheme

initial: Parental DS points to DNSKEY_S_1. All RRsets in the zone are signed with DNSKEY_S_1.

new DNSKEY: A new key (DNSKEY_S_2) is introduced, and all the RRsets are signed with both DNSKEY_S_1 and DNSKEY_S_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_S_1 has been expired), the parental DS record can be changed.

DNSKEY removal: After the DS RRset containing DS_S_1 has expired from distant caches, DNSKEY_S_1 can be removed from the DNSKEY RRset.

In this first variant, the new signatures and new public key are added to the zone. Once they are propagated, the DS at the parent is switched. If the old DS has expired from the caches, the old signatures and old public key can be removed from the zone.

This rollover has the drawback that it introduces double signatures over all data of the zone. Taking these zone size considerations into account, it is possible to not introduce the signatures made with DNSKEY_S_2 at the "new DNSKEY" step. Instead, signatures of DNSKEY_S_1 are replaced with signatures of DNSKEY_S_2 in an additional stage between the "DS change" and "DNSKEY removal" step: After the DS RRset containing DS_S_1 has expired from distant caches, the signatures can be swapped. Only after the new signatures made with DNSKEY_S_2 have been propagated can the old public key DNSKEY_S_1 be removed from the DNSKEY RRset.

The second variant of the Single-Type Signing Scheme Key rollover is the Double-DS rollover. In this variant, one introduces a new DNSKEY into the key set and submits the new DS to the parent. The new key is not yet used to sign RRsets. The signatures made with DNSKEY_S_1 are replaced with signatures made with DNSKEY_S_2 at the moment that DNSKEY_S_2 and DS_S_2 have been propagated.

```
 -----------------------------------------------------------------------
   initial            new DS         new RRSIG         DS removal
 -----------------------------------------------------------------------
 Parent:
   SOA_0              SOA_1 -------------------------> SOA_2
   RRSIG_par(SOA)     RRSIG_par(SOA) ----------------> RRSIG_par(SOA)
   DS_S_1             DS_S_1 ------------------------>
                      DS_S_2 ------------------------> DS_S_2
   RRSIG_par(DS)      RRSIG_par(DS) -----------------> RRSIG_par(DS)

 Child:
   SOA_0              SOA_1          SOA_2             SOA_3
   RRSIG_S_1(SOA)     RRSIG_S_1(SOA) RRSIG_S_2(SOA)    RRSIG_S_2(SOA)

   DNSKEY_S_1         DNSKEY_S_1     DNSKEY_S_1
                      DNSKEY_S_2     DNSKEY_S_2        DNSKEY_S_2
   RRSIG_S_1 (DNSKEY)                RRSIG_S_2(DNSKEY) RRSIG_S_2(DNSKEY)
 -----------------------------------------------------------------------
```

Figure 7: Stages of Deployment for a Double-DS Rollover in a Single-Type Signing Scheme

4.1.4. Algorithm Rollovers

A special class of key rollovers is the one needed for a change of signature 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 later.

There exist both a conservative and a liberal approach for algorithm rollover. This has to do with Section 2.2 of RFC 4035 [RFC4035]:

> 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 strictly, meaning that it expects that every RRset has a valid signature for every algorithm signaled by the zone apex DNSKEY RRset, including RRsets in caches. 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 ("Zone Signing") of RFC 4035.

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 KSK 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 to 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 described in this section will explain the stages of deployment and will assume that the conservative approach is used.

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 KSK rollover.

When removing an old algorithm, the DS for the algorithm should be removed from the parent zone first, followed by the DNSKEY and the signatures (in the child zone).

Figure 8 describes the steps.

```
   ----------------------------------------------------------------
    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_Z_10(SOA)      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_K_2
    DNSKEY_Z_10          DNSKEY_Z_10          DNSKEY_Z_10
                                              DNSKEY_Z_11
    RRSIG_K_1(DNSKEY)    RRSIG_K_1(DNSKEY)    RRSIG_K_1(DNSKEY)
                                              RRSIG_K_2(DNSKEY)

   ----------------------------------------------------------------
    new DS               DNSKEY removal       RRSIGs removal
   ----------------------------------------------------------------
   Parent:
    SOA_1 ------------------------------------------------------->
    RRSIG_par(SOA) ---------------------------------------------->
    DS_K_2 ------------------------------------------------------>
    RRSIG_par(DS_K_2) ------------------------------------------->

   Child:
    -------------------> SOA_3                SOA_4
    -------------------> RRSIG_Z_10(SOA)
    -------------------> RRSIG_Z_11(SOA)      RRSIG_Z_11(SOA)

    ------------------->
    -------------------> DNSKEY_K_2           DNSKEY_K_2
    ------------------->
    -------------------> DNSKEY_Z_11          DNSKEY_Z_11
    ------------------->
    -------------------> RRSIG_K_2(DNSKEY)    RRSIG_K_2(DNSKEY)
   ----------------------------------------------------------------
```

Figure 8: Stages of Deployment during an Algorithm Rollover

initial: Describes the state of the zone before any transition is done. The number of the keys may vary, but all keys (in DNSKEY records) for the zone use the same algorithm.

new RRSIGs: The signatures made with the new key over all records in the zone are added, but the key itself is not. This step is needed to propagate the signatures created with the new algorithm to the caches. If this is not done, it is possible for a resolver to retrieve the new DNSKEY RRset (containing the new algorithm) but to have RRsets in its cache with signatures created by the old DNSKEY RRset (i.e., without the new algorithm).

The RRSIG for the DNSKEY RRset does not need to be pre-published (since these records will travel together) and does not need special processing in order to keep them synchronized.

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

new DS: After the cache data for the old DNSKEY RRset 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 old DS RRset has expired, the old algorithm can be removed. This time, the old key needs to be removed first, before removing the old signatures.

RRSIGs removal: After the cache data for the old DNSKEY RRset has expired, the old 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 there is no differentiation between ZSKs and KSKs (Section 4.1.4.1).

2: RFC 5011 Algorithm rollover: when trust anchors can track the roll via RFC 5011 style rollover (Section 4.1.4.2).

3: 1 and 2 combined: when a Single-Type Signing Scheme Algorithm rollover is performed RFC 5011 style (Section 4.1.4.3).

In addition to the narrative below, these special cases are represented in Figures 12, 13, and 14 in Appendix C.

4.1.4.1. Single-Type Signing Scheme Algorithm Rollover

If one key is used that acts as both ZSK and KSK, the same scheme and figure as above (Figure 8 in Section 4.1.4) applies, whereby all DNSKEY_Z_* records from the table are removed and all RRSIG_Z_* are replaced with RRSIG_S_*. All DNSKEY_K_* records are replaced with DNSKEY_S_*, and all RRSIG_K_* records are replaced with RRSIG_S_*. 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.

This is shown in Figure 12 in Appendix C.

4.1.4.2. Algorithm Rollover, RFC 5011 Style

Trust anchor algorithm rollover is almost as simple as a regular RFC 5011-based rollover. However, the old trust anchor must be revoked before it is removed from the zone.

The timeline (see Figure 13 in Appendix C) is similar to that of Figure 8 above, but after the "new DS" step, an additional step is required where the DNSKEY is revoked. The details of this step ("revoke DNSKEY") are shown in Figure 9 below.

```
---------------------------------
  revoke DNSKEY
---------------------------------
Parent:
  ------------------------------>
  ------------------------------>
  ------------------------------>
  ------------------------------>

Child:
  SOA_3
  RRSIG_Z_10(SOA)
  RRSIG_Z_11(SOA)

  DNSKEY_K_1_REVOKED
  DNSKEY_K_2

  DNSKEY_Z_11
  RRSIG_K_1(DNSKEY)
  RRSIG_K_2(DNSKEY)
---------------------------------
```

Figure 9: The Revoke DNSKEY State That Is Added to an Algorithm Rollover when RFC 5011 Is in Use

There is one exception to the requirement from RFC 4035 quoted in Section 4.1.4 above: While all zone data must be signed with an unrevoked key, it is permissible to sign the key set with a revoked key. The somewhat esoteric argument is as follows:

Resolvers that do not understand the RFC 5011 REVOKE flag will handle DNSKEY_K_1_REVOKED the same as if it were 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 RFC 5011 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 is, we only need RRSIG_K_2(DNSKEY) to authenticate the DNSKEY RRset, and we are still compliant with Section 2.2 of RFC 4035: There must be an RRSIG for each RRset using at least one DNSKEY of each algorithm in the zone apex DNSKEY RRset.

4.1.4.3. Single Signing Type Algorithm Rollover, RFC 5011 Style

If a decision is made to perform an RFC 5011 style rollover with a Single Signing Scheme key, it should be noted that Section 2.1 of 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 once DNSKEY_S_1 is revoked, it cannot be used to validate its signatures over non-DNSKEY RRsets. Thus, those RRsets should be signed with a shadow key, DNSKEY_Z_10, during the algorithm rollover. The shadow key can be removed at the same time the revoked DNSKEY_S_1 is removed from the zone. In other words, the zone must temporarily fall back to a KSK/ZSK split model during the rollover.

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. 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 a candidate for revocation. That key can than temporarily act as a ZSK during the algorithm rollover.

As with algorithm rollover RFC 5011 style, while all zone data must be signed with an unrevoked key, it is permissible to sign the key set with a revoked key using the same esoteric argument given in Section 4.1.4.2.

The lesson of all of this is that a Single-Type Signing Scheme algorithm rollover using RFC 5011 is as complicated as the name of the rollover implies: Reverting to a split-key scheme for the duration of the rollover may be preferable.

4.1.4.4. 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 RRset has populated distant caches, at the end of the "new DNSKEY" stage. 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 the 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 during the "new DS" step: increasing the serial number, introducing the NSEC3PARAM record to signal that NSEC3-authenticated data related to denial of existence should be served, and re-signing the zone.

In summary, 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. The procedures are also listed in Sections 10.4 and 10.5 of RFC 5155 [RFC5155].

4.1.5. 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 the 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. It is advisable to have a documented procedure ready for those times when a key compromise is suspected or confirmed.

When the private material of one of a zone’s keys is compromised, it can be used by an attacker 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, and

- as long as the DS RR in the parent zone points to the (compromised) key signing the DNSKEY RRset, and

- as long as the (compromised) 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 a zone’s compromised key exists, your namespace is vulnerable to abuse by anyone who has obtained illegitimate possession of the key. Zone administrators have to make a decision as to whether the abuse of the compromised key is worse than having data in caches that cannot be validated. If the zone administrator 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 compromised KSK can be used to sign the key set of an attacker’s version of the zone. That zone could be used to poison the DNS.

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

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, although the chain of trust of this particular key is broken.

Note that an attacker’s version of the 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 version of the zone to appear as valid and the original zone as Bogus. Therefore, we advise administrators not to remove the KSK before the parent has a DS record for the new KSK in place.

4.2.1.1. Emergency Key Rollover Keeping the Chain of Trust Intact

If it is desired to perform an emergency key rollover in a manner that keeps the chain of trust intact, 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. This means ensuring 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 the DNSKEY RRset will expire from caches sooner.

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 DSs 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 at 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 period.

An additional danger of a key compromise is that the compromised key could be used to facilitate a legitimate-looking DNSKEY/DS rollover and/or name server 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 to authenticate communication with the parent.

4.2.1.2. Emergency Key Rollover Breaking the Chain of Trust

There are two methods to perform an emergency key rollover in a manner that breaks 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 possession 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. After the DS has expired from distant caches, the keys and signatures are removed from the child zone, new keys and signatures are introduced, and finally, a new DS is submitted to the parent.

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 still be 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 verify the new key. New keys should be authenticated out-of-band, for example, through the use of an announcement website that is secured using Transport Layer Security (TLS) [RFC5246].

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 normal DNS operation, to prepare stand-by keys you need to:

- Generate a stand-by ZSK and KSK. Store them safely in a location different than the place where the currently used ZSK and KSK are held.

- Pre-publish the DNSKEY RR of the stand-by ZSK in the zone.

- Pre-publish the DS of the stand-by KSK in the parent zone.

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

- Set up your DNS operations and introduce the stand-by KSK into the zone.

- Post-publish the disabled ZSK and sign the zone with the stand-by keys.

- After some time, when the new signatures have been propagated, the old keys, old signatures, and the old DS can be removed.

- Generate a new stand-by key set 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-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 [RFC3375] for a comprehensive definition). The key material is to be passed from the DNS operator to the parent via a registrar, where both the DNS operator and registrar are selected by the registrant and might be different organizations. 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 the 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 [RFC4035] to select the proper key(s) 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 that the child is secure.

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

With some types 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 DSs. 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 suggest that registry systems should be able to store DS RRs, even if they also store DNSKEYs (see also "DNSSEC Trust Anchor Configuration and Maintenance" [DNSSEC-TRUST-ANCHOR]).

The storage considerations also relate to the design of the customer interface and the method by which data is transferred between the 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) [RFC5910], 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 to move 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 nonexistent DNSKEY RR. Security lameness may occur temporarily during a Double-DS rollover scheme. However, care should be taken that not all DS RRs are pointing to a nonexistent DNSKEY RR, which will 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 the 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 that 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 the 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 zone administrator unavailability during weekends shows the need for DS signature validity periods longer than two days. Just like any signature validity period, we suggest an absolute minimum for the 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 period increases the operational risk for the parent. Therefore, the parent may have a policy to use a signature validity period 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 several months.

In addition to the signature validity period, which sets a lower bound on the number of times the zone administrator will need to sign the zone data and an upper bound on the time that 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 relationship 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 [RFC3375]. Registrants may outsource the maintenance of their DNS system, including the maintenance of DNSSEC key material, to the registrar or to another third party, referred to here as the 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 the losing operator has 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 the 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, the change could be made 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 keys and independently generate a signature over those key sets that they combine and both publish in their copy of the zone. Once that is done, they can use their own private keys to sign any of their zone content during the transfer.

```
    ------------------------------------------------------------
    initial            |        pre-publish                    |
    ------------------------------------------------------------
    Parent:
     NS_A                            NS_A
     DS_A                            DS_A
    ------------------------------------------------------------
    Child at A:            Child at A:        Child at B:
     SOA_A0                 SOA_A1             SOA_B0
     RRSIG_Z_A(SOA)         RRSIG_Z_A(SOA)     RRSIG_Z_B(SOA)

     NS_A                   NS_A               NS_B
     RRSIG_Z_A(NS)          NS_B               RRSIG_Z_B(NS)
                            RRSIG_Z_A(NS)

     DNSKEY_Z_A             DNSKEY_Z_A         DNSKEY_Z_A
                            DNSKEY_Z_B         DNSKEY_Z_B
     DNSKEY_K_A             DNSKEY_K_A         DNSKEY_K_A
                            DNSKEY_K_B         DNSKEY_K_B
     RRSIG_K_A(DNSKEY)      RRSIG_K_A(DNSKEY)  RRSIG_K_A(DNSKEY)
                            RRSIG_K_B(DNSKEY)  RRSIG_K_B(DNSKEY)
    ------------------------------------------------------------

    ------------------------------------------------------------
          re-delegation                |   post-migration      |
    ------------------------------------------------------------
    Parent:
              NS_B                           NS_B
              DS_B                           DS_B
    ------------------------------------------------------------
    Child at A:        Child at B:           Child at B:

     SOA_A1             SOA_B0                SOA_B1
     RRSIG_Z_A(SOA)     RRSIG_Z_B(SOA)        RRSIG_Z_B(SOA)

     NS_A               NS_B                  NS_B
     NS_B               RRSIG_Z_B(NS)         RRSIG_Z_B(NS)
     RRSIG_Z_A(NS)

     DNSKEY_Z_A         DNSKEY_Z_A
     DNSKEY_Z_B         DNSKEY_Z_B            DNSKEY_Z_B
     DNSKEY_K_A         DNSKEY_K_A
     DNSKEY_K_B         DNSKEY_K_B            DNSKEY_K_B
     RRSIG_K_A(DNSKEY)  RRSIG_K_A(DNSKEY)
     RRSIG_K_B(DNSKEY)  RRSIG_K_B(DNSKEY)     RRSIG_K_B(DNSKEY)
    ------------------------------------------------------------
```

Figure 10: 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, and 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.

The zone is initially delegated from the parent to the name servers of operator A. Operator A uses his own ZSK and KSK to sign the zone. The cooperating operator A will pre-publish the new NS record and the ZSK and KSK of operator B, including the RRSIG over the DNSKEY RRset generated by the KSK of operator B. Operator B needs to publish the same DNSKEY RRset. When that DNSKEY RRset has populated the caches, the re-delegation can be made, which involves adjusting the NS and DS records in the parent zone to point to operator B. And after all DNSSEC records related to operator A have expired from the caches, operator B can stop publishing the keys and signatures belonging to operator A, and vice versa.

The requirement to exchange signatures has a couple of drawbacks. It requires more operational overhead, because not only do the operators have to exchange public keys but they also have to exchange the signatures of the new DNSKEY RRset. This drawback does not exist if the Double-Signature KSK rollover is replaced with a Double-DS KSK rollover. See Figure 15 in Appendix D for the diagram.

Thus, if the registry and registrars allow DS records to be published that do not point to a published DNSKEY in the child zone, the Double-DS KSK rollover is preferred (see Figure 5), in combination with the Pre-Publish ZSK rollover. This does not require sharing the KSK signatures between the operators, but both operators still have to publish each other’s ZSKs.

4.3.5.2. Non-Cooperating DNS Operators

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

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 re-delegation from the losing operator to the gaining operator has taken place. One could imagine a rollover scenario where the gaining operator takes a copy of 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 re-delegation took place, the NS RRset has by definition changed, so such a 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 the losing operator's RRSIGs have been copied.

The only viable operation for the registrant is to have his zone go Insecure for the duration of the change. The registry should be asked to remove the DS RR pointing to the losing operator's DNSKEY and to change the NS RRset to point to the gaining operator. Once this has propagated through the DNS, the registry should be asked to insert the DS record pointing to the (newly signed) zone at operator B.

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

- TTL sanity checking, as described in RFC 2308 [RFC2308], will limit the impact of 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 continuing to use a name server that might be un-delegated by the parent.

- Limiting the time that 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 reduce the effects of the problem of non-cooperating registrars.

However, there is no operational methodology to work around this business issue, and proper contractual relationships between all involved parties seem 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 are referenced by that zone also change operators, possibly involving some downtime.

In any case, to minimize such problems, the classic configuration is to have relatively short TTLs 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 that has elapsed after a slave server synchronized with a master server. The TTL value and the SOA RR minimum TTL parameter [RFC2308] are used to determine how long a forwarder should cache data (or negative responses) 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 "DNSSEC Key Timing Considerations" [DNSSEC-KEY-TIMING].

4.4.1. Time Considerations

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

- We suggest that the Maximum Zone TTL value of your zone data be smaller than your signature validity period.

    If the TTL duration was similar to that of the signature validity period, then all RRsets fetched during the validity period would be cached until the signature expiration time. Section 8.1 of RFC 4033 [RFC4033] 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, the query load on authoritative servers would peak at the signature expiration time, as this is also the time at which records simultaneously expire from caches.

    Having a TTL that is at least a few times smaller than your signature validity period avoids query load peaks.

- We suggest that the signature publication period end at least one Maximum Zone TTL duration (but preferably a minimum of 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 the 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 re-signing operation, and those signatures that are within a so-called Refresh Period from signature expiration are recreated. Also see Section 4.4.2 below.

In the 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 that the signatures will survive at least a (long) weekend in case of such operational havoc. This is called the Refresh Period (see Section 4.4.2).

- We suggest that the Minimum Zone TTL be long enough to both fetch and verify all the RRs in the trust chain. In workshop environments, it has been demonstrated [NIST-Workshop] 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 name servers. Data at delegation points, DS, DNSKEY, and RRSIG RRs benefits from caching. The TTL on those should be relatively long. Data at the leaves in the DNS tree has less impact on recursive name servers.

- 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-secured zone, then it may 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 that the SOA expiration timer be 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 signatures time out.

We also suggest that operators of name servers 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 secondaries expire the zone? How quickly can the administrators of the secondary servers be reached to load a valid zone? These questions are not DNSSEC-specific but may influence the choice of your signature validity periods.

4.4.2. Signature Validity 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 those given in Section 3.2.2 play a role: When a zone is re-signed with some regularity, then zone administrators remain conscious of 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: At what time will a failure be noticed, and at what time is action expected to be taken? By answering these questions, availability of zone administrators during (long) weekends or time taken to access backup media can be taken into account. The result could easily suggest a minimum signature validity period of a few days.

Note, however, that 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 within a given amount of time (the Refresh Period) of expiration. The Re-Sign Period must be smaller than the Refresh Period in order for zone data to be signed in a timely fashion.

If an operational problem occurs during re-signing, 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.

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.

In other words, the minimum signature validity period 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 Re-Sign Period, minus the maximum jitter sets the time in which operational havoc can be resolved.

The relationship between signature times is illustrated in Figure 11.

```
   Inception          Signing                                 Expiration
   time               time                                    time
   |                  |                                 |     |     |
   |------------------|---------------------------------|.....|.....|
   |                  |                                 |     |     |
                                                          +/-jitter

   | Inception offset |                                       |
   |<---------------->|            Validity Period            |
   |               |<---------------------------------------->|


   Inception          Signing Reuse   Reuse   Reuse   New     Expiration
   time               time                            RRSIG   time
   |                  |       |       |       |       |       |
   |------------------|-------------------------------|-------|
   |                  |       |       |       |       |       |
                       <-----> <-----> <-----> <----->
                     Re-Sign Period

                                                |   Refresh   |
                                                |<----------->|
                                                |   Period    |
```

Figure 11: Signature Timing Parameters

Note that in the figure the validity of the signature starts shortly before the signing time. That is done to deal with validators that might have some clock skew. This is called the inception offset, and it should be chosen so that false negatives are minimized to a reasonable level.

4.4.2.3. Differentiation between RRsets

It is possible to vary signature validity periods between signatures over different RRsets 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) should be the leading factor in determining the signature validity period, since the TTLs on the data itself are still 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 period protects the child against replay attacks in the case where 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 period 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" Types

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 nonexistent names that lie between the names that do exist.

There are two mechanisms to provide authenticated proof of nonexistence 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 nonexistence 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 that DNS data is accessible through query mechanisms, for some zone administrators this behavior is undesirable for policy, regulatory, or other reasons.

Furthermore, NSEC requires a signature over every RR in the zone file, thereby ensuring that any denial of existence is cryptographically signed. However, in a large zone file 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 updates (a typical example might be a TLD operator with few registrants using secure delegations). NSEC3 allows intervals between two secure delegations to "opt out", in which case they may contain one or 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 name. To increase the workload required to guess entries in the zone, the number of hashing iterations 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 (you can still derive the size of the zone, which RRTYPEs are in there, etc.).

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 in-addr.arpa., ip6.arpa., and e164.arpa., can be trivially enumerated using ordinary DNS properties, while for small zones that only contain records in the apex of the zone and a few common names such as "www" or "mail", guessing zone content and proving completeness is also trivial when using NSEC3. In these cases, the use of NSEC is preferred to ease the work required by signers and validating resolvers.

For large zones where there is an implication of "not readily available" names, such as those where one has to sign a non-disclosure agreement before obtaining it, NSEC3 is preferred. The second reason to consider NSEC3 is "Opt-Out", which can reduce the number of NSEC3 records required. This is discussed further below (Section 5.3.4).

5.3. NSEC3 Parameters

NSEC3 is controlled by a number of parameters, some of which can be varied: This section discusses the choice of those parameters.

5.3.1. NSEC3 Algorithm

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

At the time of this writing, there is only one NSEC3 hash algorithm defined. [RFC5155] specifically states: "When specifying a new hash algorithm for use with NSEC3, 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 that a pre-calculated dictionary attack could be performed in order to assess whether or not certain domain names exist within a zone. Two mechanisms are introduced in the NSEC3 specification to increase the costs of such dictionary attacks: iterations and salt.

The iterations parameter defines the number of additional times the hash function has been performed. A higher value results in greater resiliency against dictionary attacks, at a higher computational cost for both the server and resolver.

RFC 5155 Section 10.3 [RFC5155] considers the tradeoffs between incurring cost during the signing process and imposing costs to the validating name server, 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 name servers would be halved, regardless of key size [NSEC3-HASH-PERF].

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 administrator would cause an attacker to lose all pre-calculated work for that zone.

There must be a complete NSEC3 chain using the same salt value, that matches the salt value in the NSEC3PARAM record. NSEC3 salt changes do not need special rollover procedures. Since changing the salt requires that all the NSEC3 records be regenerated and thus requires generating new RRSIGs over these NSEC3 records, it makes sense to align the change of the salt with a change of the Zone Signing Key, as that process in itself already usually requires that all RRSIGs be regenerated. 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.

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 proof that DNSSEC is not available for the delegations within the span. 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 insecure delegations. This consideration typically holds for large TLDs and similar zones; in most other circumstances, Opt-Out should not be deployed. Further considerations can be found in Section 12.2 of RFC 5155 [RFC5155].

6. Security Considerations

DNSSEC adds data origin authentication and data integrity to the DNS, using digital signatures over Resource Record sets. DNSSEC does not protect against denial-of-service attacks, nor does it provide confidentiality. For more general security considerations related to DNSSEC, please see RFC 4033 [RFC4033], RFC 4034 [RFC4034], and RFC 4035 [RFC4035].

This document tries to assess the operational considerations to maintain a stable and secure DNSSEC service. When performing key rollovers, it is important to keep in mind that it takes time for the data to be propagated to the verifying clients. It is also important to note that this data may be cached. Not taking into account the 'data propagation' properties in the DNS may cause validation failures, because cached data may mismatch data fetched from the authoritative servers; this will make secured zones unavailable to security-aware resolvers.

7. Acknowledgments

Significant parts of the text of this document are copied from RFC 4641 [RFC4641]. That document was edited by Olaf Kolkman and Miek Gieben. Other people that contributed or were 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, Lindy Foster, and O. Courtay.

For this version of the document, we would like to acknowledge people who were actively involved in the compilation of the document. In random order: Mark Andrews, Patrik Faltstrom, Tony Finch, Alfred Hoenes, Bill Manning, Scott Rose, Wouter Wijngaards, Antoin Verschuren, Marc Lampo, George Barwood, Sebastian Castro, Suresh Krishnaswamy, Eric Rescorla, Stephen Morris, Olafur Gudmundsson, Ondrej Sury, and Rickard Bellgrim.

8. Contributors

Significant contributions to this document were from:

Paul Hoffman, who contributed on the choice of cryptographic parameters and addressing some of the trust anchor issues;

Jelte Jansen, who provided the initial text in Section 4.1.4;

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

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

9. References

9.1. Normative References

[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.

[RFC4509] Hardaker, W., "Use of SHA-256 in DNSSEC Delegation Signer (DS) Resource Records (RRs)", RFC 4509, May 2006.

[RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, March 2008.

[RFC5702] Jansen, J., "Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC", RFC 5702, October 2009.

9.2. Informative References

[RFC1995] Ohta, M., "Incremental Zone Transfer in DNS", RFC 1995, August 1996.

[RFC1996] Vixie, P., "A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)", RFC 1996, August 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, March 1998.

[RFC3007] Wellington, B., "Secure Domain Name System (DNS) Dynamic Update", RFC 3007, November 2000.

[RFC3375] Hollenbeck, S., "Generic Registry-Registrar Protocol Requirements", RFC 3375, September 2002.

[RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", BCP 86, RFC 3766, April 2004.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

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

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.

[RFC5011] StJohns, M., "Automated Updates of DNS Security (DNSSEC) Trust Anchors", RFC 5011, September 2007.

[RFC5910] Gould, J. and S. Hollenbeck, "Domain Name System (DNS) Security Extensions Mapping for the Extensible Provisioning Protocol (EPP)", RFC 5910, May 2010.

[RFC5933] Dolmatov, V., Chuprina, A., and I. Ustinov, "Use of GOST Signature Algorithms in DNSKEY and RRSIG Resource Records for DNSSEC", RFC 5933, July 2010.

[RFC6605] Hoffman, P. and W. Wijngaards, "Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC", RFC 6605, April 2012.

[NIST-Workshop]
Rose, S., "NIST DNSSEC workshop notes", July 2001, <http://www.ietf.org/mail-archive/web/dnsop/current/msg01020.html>.

[NIST-SP-800-90A]
Barker, E. and J. Kelsey, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators", NIST Special Publication 800-90A, January 2012.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[DNSSEC-KEY-TIMING]
Morris, S., Ihren, J., and J. Dickinson, "DNSSEC Key Timing Considerations", Work in Progress, July 2012.

[DNSSEC-DPS]
Ljunggren, F., Eklund Lowinder, AM., and T. Okubo, "A Framework for DNSSEC Policies and DNSSEC Practice Statements", Work in Progress, November 2012.

[DNSSEC-TRUST-ANCHOR]
Larson, M. and O. Gudmundsson, "DNSSEC Trust Anchor Configuration and Maintenance", Work in Progress, October 2010.

[NSEC3-HASH-PERF]
Schaeffer, Y., "NSEC3 Hash Performance", NLnet Labs document 2010-002, 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 RFC 4033 [RFC4033]. An RRset in DNSSEC is marked "Bogus" when a signature of an RRset does not validate against a DNSKEY.

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.

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 the use of the DNSKEY Resource Record (DNSKEY RR). Private keys should remain private.

Refresh Period: The period before the expiration time of the signature, during which the signature is refreshed by the signer.

Re-Sign Period: This refers to the frequency with which a signing pass on the zone is performed. The Re-Sign Period defines when the zone is exposed to the signer. And on the signer, not all signatures in the zone have to be regenerated: That depends on the Refresh Period.

Secure Entry Point (SEP) key: A KSK that has a DS record in the parent zone pointing to it or that is configured as a trust anchor. Although not required by the protocol, we suggest that the SEP flag [RFC4034] be 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: A random variation in the signature validity period of RRSIGs in a zone to prevent all of them from expiring at the same time.

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.

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

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

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

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.

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 Key Signing Key, Z for Zone Signing Key, and S when there is no distinction made between KSKs 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.

RRsets ignored: If the signatures of non-DNSKEY RRsets have the same parameters as the SOA, then 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.

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

Signature notation: Signatures are denoted as RRSIG_x_y(type), which means that the RRset with the specific RRTYPE ’type’ is signed with DNSKEY_x_y. Signatures in the parent zone are denoted as RRSIG_par(type).

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

DS representation: DSs are represented as DS_x_y, where x and y are identifiers similar to the key notation: x is an identifier for the type of key the DS record refers to; y is the ’key id’ of the key it refers to.

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".

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_K_14.

Appendix C. Transition Figures for Special Cases of Algorithm Rollovers

The figures in this appendix complement and illustrate the special cases of algorithm rollovers as described in Section 4.1.4.

```
   ----------------------------------------------------------------
    initial              new RRSIGs           new DNSKEY
   ----------------------------------------------------------------
   Parent:
    SOA_0 -------------------------------------------------------->
    RRSIG_par(SOA) ----------------------------------------------->
    DS_S_1 ------------------------------------------------------->
    RRSIG_par(DS_S_1) -------------------------------------------->

   Child:
    SOA_0                SOA_1                SOA_2
    RRSIG_S_1(SOA)       RRSIG_S_1(SOA)       RRSIG_S_1(SOA)
                         RRSIG_S_2(SOA)       RRSIG_S_2(SOA)

    DNSKEY_S_1           DNSKEY_S_1           DNSKEY_S_1
                                              DNSKEY_S_2
    RRSIG_S_1(DNSKEY)    RRSIG_S_1(DNSKEY)    RRSIG_S_1(DNSKEY)
                         RRSIG_S_2(DNSKEY)    RRSIG_S_2(DNSKEY)

   ----------------------------------------------------------------
    new DS               DNSKEY removal       RRSIGs removal
   ----------------------------------------------------------------
   Parent:
    SOA_1 ------------------------------------------------------->
    RRSIG_par(SOA) ---------------------------------------------->
    DS_S_2 ------------------------------------------------------>
    RRSIG_par(DS_S_2) ------------------------------------------->

   Child:
    -------------------> SOA_3                SOA_4
    -------------------> RRSIG_S_1(SOA)
    -------------------> RRSIG_S_2(SOA)       RRSIG_S_2(SOA)

    ------------------->
    -------------------> DNSKEY_S_2           DNSKEY_S_2
    -------------------> RRSIG_S_1(DNSKEY)
    -------------------> RRSIG_S_2(DNSKEY)    RRSIG_S_2(DNSKEY)
   ----------------------------------------------------------------
```

Figure 12: Single-Type Signing Scheme Algorithm Roll

Also see Section 4.1.4.1.

```
   ----------------------------------------------------------------
    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_Z_1(SOA)       RRSIG_Z_1(SOA)       RRSIG_Z_1(SOA)
                         RRSIG_Z_2(SOA)       RRSIG_Z_2(SOA)

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

   ----------------------------------------------------------------
    new DS               revoke DNSKEY        DNSKEY removal
   ----------------------------------------------------------------
   Parent:
    SOA_1 ------------------------------------------------------->
    RRSIG_par(SOA) ---------------------------------------------->
    DS_K_2 ------------------------------------------------------>
    RRSIG_par(DS_K_2) ------------------------------------------->

   Child:
    -------------------> SOA_3                SOA_4
    -------------------> RRSIG_Z_1(SOA)       RRSIG_Z_1(SOA)
    -------------------> RRSIG_Z_2(SOA)       RRSIG_Z_2(SOA)

    -------------------> DNSKEY_K_1_REVOKED
    -------------------> DNSKEY_K_2           DNSKEY_K_2
    ------------------->
    -------------------> DNSKEY_Z_2           DNSKEY_Z_2
    -------------------> RRSIG_K_1(DNSKEY)
    -------------------> RRSIG_K_2(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)
   ----------------------------------------------------------------
```

Figure 13: RFC 5011 Style Algorithm Roll

Also see Section 4.1.4.2.

```
   ----------------------------------------------------------------
    initial              new RRSIGs           new DNSKEY
   ----------------------------------------------------------------
   Parent:
    SOA_0 -------------------------------------------------------->
    RRSIG_par(SOA) ----------------------------------------------->
    DS_S_1 ------------------------------------------------------->
    RRSIG_par(DS_S_1) -------------------------------------------->

   Child:
    SOA_0                SOA_1                SOA_2
    RRSIG_S_1(SOA)
    RRSIG_Z_10(SOA)      RRSIG_Z_10(SOA)      RRSIG_Z_10(SOA)
                         RRSIG_S_2(SOA)       RRSIG_S_2(SOA)

    DNSKEY_S_1           DNSKEY_S_1           DNSKEY_S_1
    DNSKEY_Z_10          DNSKEY_Z_10          DNSKEY_Z_10
                                              DNSKEY_S_2
    RRSIG_S_1(DNSKEY)    RRSIG_S_1(DNSKEY)    RRSIG_S_1(DNSKEY)
                         RRSIG_S_2(DNSKEY)    RRSIG_S_2(DNSKEY)
   ----------------------------------------------------------------
    new DS               revoke DNSKEY        DNSKEY removal
   ----------------------------------------------------------------
   Parent:
    SOA_1 ------------------------------------------------------->
    RRSIG_par(SOA) ---------------------------------------------->
    DS_S_2 ------------------------------------------------------>
    RRSIG_par(DS_S_2) ------------------------------------------->

   Child:
    -------------------> SOA_3                SOA_4

    -------------------> RRSIG_Z_10(SOA)
    -------------------> RRSIG_S_2(SOA)       RRSIG_S_2(SOA)

    -------------------> DNSKEY_S_1_REVOKED
    -------------------> DNSKEY_Z_10
    -------------------> DNSKEY_S_2           DNSKEY_S_2
    -------------------> RRSIG_S_1(DNSKEY)    RRSIG_S_1(DNSKEY)
    -------------------> RRSIG_S_2(DNSKEY)    RRSIG_S_2(DNSKEY)

   ----------------------------------------------------------------
    RRSIGs removal
   ----------------------------------------------------------------
   Parent:
    ------------------------------------->
    ------------------------------------->
    ------------------------------------->
    ------------------------------------->

   Child:
    SOA_5


    RRSIG_S_2(SOA)



    DNSKEY_S_2

    RRSIG_S_2(DNSKEY)
   ----------------------------------------------------------------
```

Figure 14: RFC 5011 Algorithm Roll in a Single-Type
Signing Scheme Environment

Also see Section 4.1.4.3.

Appendix D. Transition Figure for Changing DNS Operators

The figure in this Appendix complements and illustrates the special case of changing DNS operators as described in Section 4.3.5.1.

```
    ------------------------------------------------------------
    new DS             |        pre-publish                    |
    ------------------------------------------------------------
    Parent:
     NS_A                              NS_A
     DS_A DS_B                         DS_A DS_B
    ------------------------------------------------------------
    Child at A:           Child at A:        Child at B:
     SOA_A0                SOA_A1             SOA_B0
     RRSIG_Z_A(SOA)        RRSIG_Z_A(SOA)     RRSIG_Z_B(SOA)

     NS_A                  NS_A               NS_B
     RRSIG_Z_A(NS)         NS_B               RRSIG_Z_B(NS)
                           RRSIG_Z_A(NS)

     DNSKEY_Z_A            DNSKEY_Z_A         DNSKEY_Z_A
                           DNSKEY_Z_B         DNSKEY_Z_B
     DNSKEY_K_A            DNSKEY_K_A         DNSKEY_K_B
     RRSIG_K_A(DNSKEY)     RRSIG_K_A(DNSKEY)  RRSIG_K_A(DNSKEY)
                           RRSIG_K_B(DNSKEY)  RRSIG_K_B(DNSKEY)
    ------------------------------------------------------------

    ------------------------------------------------------------
          re-delegation            |   post-migration      |
    ------------------------------------------------------------
    Parent:
              NS_B                           NS_B
              DS_A DS_B                      DS_B
    ------------------------------------------------------------
    Child at A:        Child at B:           Child at B:

     SOA_A1             SOA_B0                SOA_B1
     RRSIG_Z_A(SOA)     RRSIG_Z_B(SOA)        RRSIG_Z_B(SOA)

     NS_A               NS_B                  NS_B
     NS_B               RRSIG_Z_B(NS)         RRSIG_Z_B(NS)
     RRSIG_Z_A(NS)

     DNSKEY_Z_A         DNSKEY_Z_A
     DNSKEY_Z_B         DNSKEY_Z_B            DNSKEY_Z_B
     DNSKEY_K_A         DNSKEY_K_B            DNSKEY_K_B
     RRSIG_K_A(DNSKEY)  RRSIG_K_B(DNSKEY)     RRSIG_K_B(DNSKEY)
    ------------------------------------------------------------
```

Figure 15: An Alternative Rollover Approach for Cooperating Operators

Appendix E. Summary of Changes from RFC 4641

This document differs from RFC 4641 [RFC4641] in the following ways:

- Addressed the errata listed on <http://www.rfc-editor.org/errata_search.php?rfc=4641>.

- Recommended RSA/SHA-256 in addition to RSA/SHA-1.

- Did a complete rewrite of Section 3.5 of RFC 4641 (Section 3.4.2 of this document), removing the table and suggesting a key size of 1024 for keys in use for less than 8 years, issued up to at least 2015.

- Removed the KSK for high-level zones consideration.

- Added text on algorithm rollover.

- Added text on changing (non-cooperating) DNS registrars.

- Did a significant rewrite of Section 3, whereby the argument is made that the timescales for rollovers are made purely on operational arguments.

- Added Section 5.

- Introduced Single-Type Signing Scheme terminology and made the arguments for the choice of a Single-Type Signing Scheme more explicit.

- Added a section about stand-by keys.

Authors' Addresses

Olaf M. Kolkman
NLnet Labs
Science Park 400
Amsterdam 1098 XH
The Netherlands

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

W. (Matthijs) Mekking
NLnet Labs
Science Park 400
Amsterdam 1098 XH
The Netherlands

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

R. (Miek) Gieben
SIDN Labs
Meander 501
Arnhem 6825 MD
The Netherlands

EMail: miek.gieben@sidn.nl
URI: http://www.sidn.nl