DNSSEC Key Maintenance Analysis

Jelte Jansen*, *NLnet Labs*
http://www.NLnetLabs.nl

NLnet Labs document 2008-002 version 1.0 September 11, 2008

*jelte@nlnetlabs.nl

File information:Id: 2008-002.tex 61 2008-09-11 14:39:31Z jelte

Contents

I Introduction **2**

II Attack Tree Analysis **3**

1 Private keys **4**
1.1 Cryptographic attack 5
1.1.1 Brute force 5
1.1.2 Algorithm weakness 5
1.1.3 Entropy weakness 6
1.2 Social attack 6
1.3 Physical attack 6
1.3.1 Theft 6
1.3.2 Destruction 6

2 Public keys **8**
2.1 Modification 8
2.1.1 Before publication 8
2.1.2 On the wire 8
2.1.3 At the parent 8
2.1.4 As trust anchor 8
2.2 Cryptographic attack 9
2.3 Attack on the publication mechanism 9

III Key management **10**

3 Staff **10**

4 Algorithm choice **10**

5 Key length and lifetime **11**

6 Key rollover **11**
6.1 Periodic rollover 11
6.2 Emergency rollover 11
6.3 ZSK Rollover 12
6.4 KSK Rollover as a child 14
6.5 KSK Rollover as a trust anchor 14
6.5.1 Automated rollover 14
6.6 Rollover of a child 14
6.7 Algorithm Rollover 14
6.7.1 Algorithm downgrade protection 15
6.7.2 Operational solution 15

6.7.3 Schematic representation of Operational solution 15

7 Private key storage **17**

8 Public key publication **19**
8.1 Through the parent 19
8.2 DLV 19
8.3 Out of band 19
8.4 In-band 20
8.5 Keys as secure entry points 20
8.6 Key updates of children 20

9 Management systems and tools **22**
9.1 Specific Key Management Software 22
9.2 DNSSEC Libraries and toolsets 22
9.3 HSM Support in Software 22
9.4 General Software Information 23

About NLnet Labs **24**

About the sponsor of this paper **24**

Legal Notice **24**

References **25**

CONTENTS

Executive Summary

The Domain Name System (DNS) is a critical infrastructure component of the Internet. Although invented in the early days of the Internet, its design is such that it manages to be scalable to the size and the dynamics of the Internet in present days. However, the immensive growth of the Internet was not foreseen, and the scalable design did not take the abuse patterns that comes with that into account.

In 2005, an extension to the Domain Name System was released that provides data integrity to the DNS. This extension is called DNSSEC. An important part of adopting DNSSEC as a domain name owner is storage and handling of cryptographic keys.

This document provides recommendations for the generation, storage and use of keys in the context of DNSSEC. It is a followup of NLnet Labs document 2006-SE-01: DNS Threat Analysis.

File information:Id: 2008-002.tex 61 2008-09-11 14:39:31Z jelte

Part I
Introduction

This document gives advice on DNSSEC (RFC4033 [2], RFC4034 [4], RFC4035 [3]) key management. We will specify and discuss an attack tree on DNSSEC keys, and possible ways to mitigate the attacks. We will also discuss more general key maintenance and provisioning systems and procedures. The scope of this document is DNSSEC public and private keys only, and we shall not discuss other DNSSEC issues.

The approach taken is a 'desk-study'. Since NLnet Labs does not have operational expertise in running a TLD or maintaining root certificates the report will not focus on 'low-level' operational aspects.

Cursory knowledge on DNS, DNSSEC, and the terms used while discussing these subjects, is presumed.

For this study we have exclusively worked with information that is openly available.

This report was created on request of and sponsored by .SE, the Internet Infrastructure Foundation.

Part II
Attack Tree Analysis

Figure 1 shows the attack tree for DNSSEC keys. On the following pages we will explain each node of this tree, and provide recommendations to protect against these attacks.

Although we mainly approach this chapter from the point of view of a malicious attacker, a number of these problems could also have non-malicious causes, such as an error, an accident, incompetence or sloppiness. We shall indicate those varieties when we encounter them.

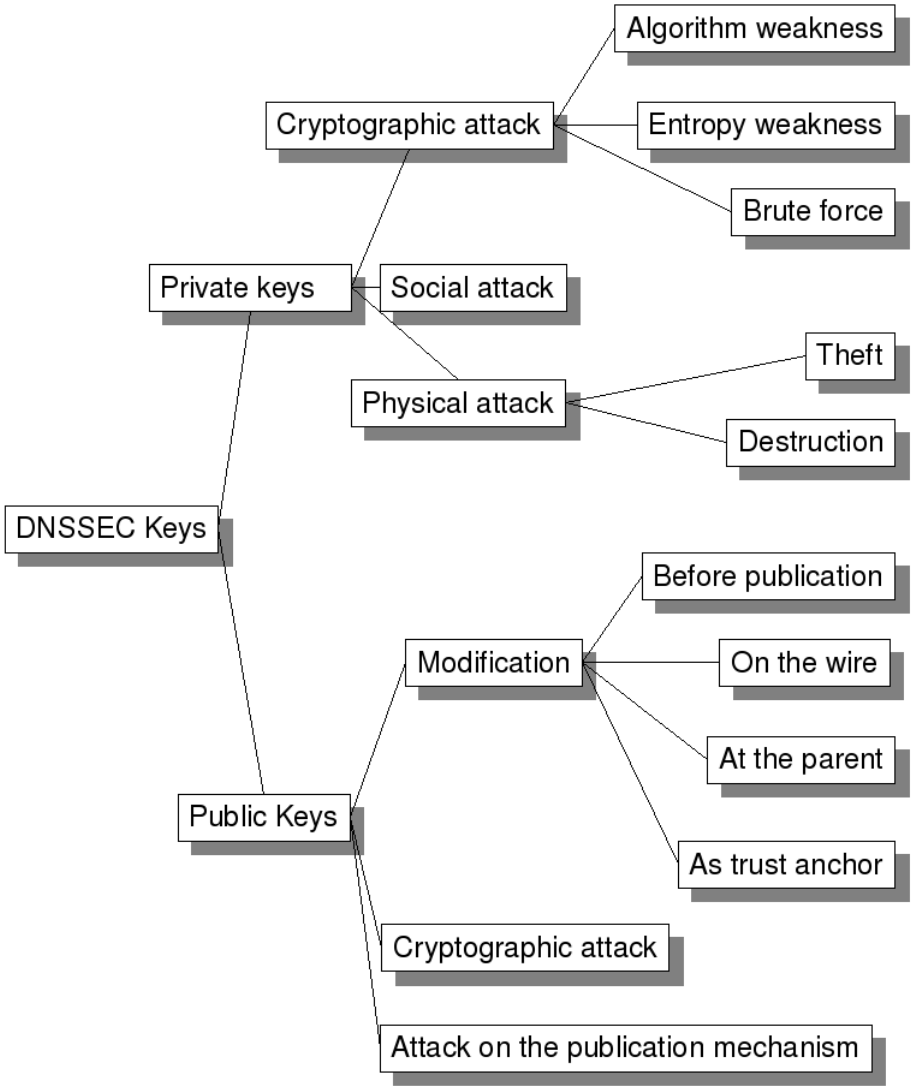

Figure 1: Attack tree

1 Private keys

The private keys are the ones used to actually sign the data, and create the RRSIG Resource Records. Of course, since the trust model is centered around these objects, they need to be protected very well.

In general there are two possible results from a successful attack on private keys:

1. The keys are rendered unusable for the administrator
2. The attacker can use the keys as if he is the administrator.

In the first case, the administrator can no longer use the keys, and new keys must be issued before the signatures with the old key expire. This is a form of denial of service.

The second case is more dangerous; if the attacker has access to the private key data, he can inject arbitrary data as if it was placed there by the owner of the zone. This reduces the level of security to that of an unsigned zone. It might be even worse if users of the data now view it as more reliable than that of an unsigned zone.

The methods by which an attacker could attack the private keys are described in the sections below.

1.1 Cryptographic attack

A cryptographic attack is targeted at the mechanics of the cryptographic algorithm used. This is what the choice of a good algorithm is meant to protect against. In this context, the goal of such an attack is to be able to create valid signatures without having access to the original private key.

1.1.1 Brute force

Brute force is simply trying out every possible key to find the one that matches. Increasing the sizes of the keys makes this form of attack harder. Administrators should strive to choose key sizes and rollover periods so that successful brute force attacks are infeasible. They should also take care not to choose too large keys, or too small rollover periods, which could cause operational problems.

DNSSEC Operational Practices (RFC4641 [9]) has a section about choosing the right key sizes, but at the time of writing, the suggested values in this document are under discussion. For a more general approach on different choices for specific cryptographic algorithms, see NIST SP 800-57: Recommendations for key management [5].

See section 5 for more information on choices for key lengths.

1.1.2 Algorithm weakness

There is a lot of research in both the field of cryptography and the field of cryptanalysis. Algorithms that are thought to be secure are discovered to contain serious flaws, and are updated or replaced by better algorithms. Obviously, when an algorithm is known as 'broken', it should not be used for the creation of new keys, and current keys using that algorithm should be phased out and replaced by others using better cryptographic algorithms as soon as possible.

Some algorithms show signs of weaknesses but are not immediately considered broken. While it is tempting to defer action, it would be wise to migrate away from these algorithms as well, so as not to have to do an emergency rollover if they should be found to be broken after all.

See section 4 for more information and recommendations on available algorithms.

1.1.3 Entropy weakness

Private keys might be very easy to reproduce if there is a shortage of entropy during their creation. The same could happen if the random generator on the used system is flawed. This is a difficult class of problems to recognize, and flaws of these kind can go unnoticed for years. For instance, there has recently been such a problem in the OpenSSL adaptation for Debian and derivatives [13].

Recommendation (1): ***On the system, or device, that is used to create the keys, make sure enough entropy can be gathered, and that this is done in a secure way.***

Recommendation (2): ***For generated keys, keep a log of the system, software versions, settings, and time of generation. That way, when an error like the above is found in any part of the system, one can know which keys are affected by it.***

1.2 Social attack

Social Engineering, obtaining sensitive data by manipulating users, can always be used where security processes are implemented. There is no technical solution to this problem. Rather, the way to mitigate this class of problems is to clearly define security procedures, train people involved, and make sure there are no shortcuts in the process.

1.3 Physical attack

Lastly, the keys have to be stored somewhere, and the storage medium can be compromised, lost, or destroyed too. A more extensive discussion on private key storage can be found in section 7.

1.3.1 Theft

The storage medium could be stolen. If it has no extra access control, stealing the medium would be enough to give the thief the ability to create correct signatures. Also, if RFC5011 [12] would be used, and there is no backup version of the same key, it cannot be revoked anymore, since the administrator no longer has access to the private key.

Recommendation (3): ***Provide a means of secure storage for important keys like KSKs. This storage medium should have its own access control.***

1.3.2 Destruction

If the only goal is a denial-of-service, the storage medium could be destroyed. The result of this would 'only' be the need of an emergency key rollover. However, if RFC5011 [12] is implemented for automatic key rollover, it would not be possible

to revoke this key when rolled. Some implementations will implement timeout values on missing keys, and remove them after a while. This is, however, not a part of the specification, and implementations could decide that a key that is not revoked is valid indefinitely.

Of course, destruction can also occur as a result of hardware failure, and measures to protect against this should be taken.

Recommendation (4): ***When choosing a storage medium, possibilities of backup of its data should be examined. Of course, these backups should have the same protection as the original, if not more.***

2 Public keys

The public part of a key pair is, as the name implies, public. In the sense of disclosure, there are no attacks here. However, the classes of attacks on public keys are mostly not on the keys themselves, but on the distribution mechanism. One could try to replace a public key with another one.

The goal of such an attack is to have the audience believe that the public key originates from the authentic source, while in fact it is not.

2.1 Modification

One type of attack on the publication mechanism is to modify or replace the published public key with the attackers own public key, so that the original signatures would be seen as bogus, and the attackers signatures would verify.

There are a few points where an attacker could possibly try to replace the public key, specified in the sections below.

2.1.1 Before publication

If, somehow, the key would be replaced in the zone, before it is signed and published, the result could have the same effect as a compromised private key. Of course, in order for the published zone to be signed correctly, the attacker would also have to replace the private key. That makes this form of attack infeasible, or at least more difficult to perform, and easy to detect.

2.1.2 On the wire

Another option would be for the attacker to actually spoof DNSKEYs in transit, as the verifying resolver is building a chain of trust. Of course, since DNSKEY RRsets are also signed, and need to match the DS record set at the parent zone, they would have to somehow also replace the signatures, or the DS record set, which means that this would, for them, just move the attacker's problem to another zone.

2.1.3 At the parent

If the attacker could somehow replace the DNSKEY or DS records at the parent with its own keys, the attackers signatures would be seen as the correct ones. This could also be seen as a problem at the parent side, and should be mitigated by having robust and safe procedures for children to pass on their DNSKEY or DS records.

2.1.4 As trust anchor

If an out of band distribution mechanism is used, care should be taken to secure this mechanism. The channel itself needs to be secure, and users need to be able to verify the keys they receive. See section 8.3 for recommendations on out of band channels.

It is also possible that a trust anchor is changed, but clients are not updated. This is in essence a denial-of-service attack by accident. See also section 8.5.

2.2 Cryptographic attack

It is possible that an algorithm is used that is so weak that the private keys could be derived from the public key. If this would be possible with anything but brute-force methods, such an algorithm would be too weak to consider using in any PKI-like system. None of the currently used algorithms show this kind of weakness, and new algorithms must pass extensive public review before they are allowed. Showing any sign of this weakness will instantly disqualify those algorithms.

2.3 Attack on the publication mechanism

Another vector of attack could be the very publication mechanism itself. An attacker could try to make users believe that a new trust anchor should be configured, providing its own public keys in the process.

Recommendation (5): ***The publication mechanism should be clearly published, in such a form that a user can be sure that a change is really issued by the administrators.***

An attacker could also try to subvert this publication mechanism itself, by issuing a fake change in the publication system itself.

Recommendation (6): ***The publication of the publication mechanism should include publication of changes in the publication mechanism itself.***

Recommendation (7): ***Every out-of-band announcement of change in either keyset or publication mechanism should be signed and verifiable by its own security mechanism, such as PGP.***

Part III
Key management

3 Staff

For important zones, the value of cryptographic keys should not be underestimated. It should be made sure that the people involved are aware of the possible results of key maintenance failures.

When key maintenance is done manually, it can be opted to make it a default policy to have more than one staff member present. Such a measure, together with training, can prevent a lot of accidental failures of critical systems.

Recommendation (8): ***Make sure that staff members are well trained and skilled, and that they take key management seriously.***

4 Algorithm choice

Currently, one can choose between the following algorithms:

- **RSA/MD5** Due to the weakness of MD5, this algorithm has the status NOT RECOMMENDED in RFC4034 Appendix A1 [4].
- **RSA/SHA-1** At the moment, this algorithm appears to be the most used. While recent cryptanalysis discoveries have made SHA-1 a little weaker, it should still be strong enough to use.
- **DSA** This is actually DSS; DSA with SHA-1. DSA can be a better choice than RSA, but be sure to read the considerations sections in RFC2536 [7].

A current Internet-Draft specifies the use of SHA-256 and SHA-512 signatures [8] with RSA. An Internet-Draft on the use of Elliptic Curve Cryptography (ECC) was also worked on, but the last draft has expired. The authors of this document do not know whether there are any plans to revive said draft [11].

When choosing one of the RSA algorithms, make sure to use a large exponent when generating keys. Usually, a Fermat prime is chosen as the exponent. F0 (3) has been a popular choice, but it was shown to generate bad keys and should not be used. F2 (17) is not recommended either. We recommend to use at least F4 (65537). While the known attacks on RSA with a low exponent are to break confidentiality [1] and not integrity, there are also known cases where an implementation error was exploited because of a low exponent [6]. Therefore we advice to choose a higher one.

When DSA is used, a thing to keep in mind is that the creation of signatures needs entropy too, and a good entropy source is needed not only for the key generator, but also for the signer. If a bad entropy source is used when creating a signature, like the Debian bug mentioned in section 1.1.3, there is an attack that can derive the key from that signature.

5 Key length and lifetime

Depending on the intended use of a key pair, and the importance of the data protected by it, different effectivity periods can be choosen. Some general guidelines can be found in NIST SP 800-57 [5].

DNSSEC Operational Practices (RFC4641 [9]) also provides recommendations on key lengths and updates. Recently, however, issues have been raised regarding the contents of this informational RFC. At the time of writing, discussions are ongoing on whether to reopen this document.

In private communication, Paul Hoffman had this to say:

> In modern usage, 1024 bit signing keys are quite sufficient for practical use for DNSSEC signing keys. The most concerted published attack to date has broken a 700ish bit key; see http://eprint.iacr.org/2007/205, particularly section 7, which is quite readable. When deciding the key length to use, it is important to consider the cost of an emergency rollover in case of a compromise; the easier the rollover, the shorter the key you need. Trust anchor keys have a higher cost to the users of key roll-overs than the cost of rolling over keys that are distributed by a parent. Regardless of that, since no one has publicly broken anything near a 1024 bit key, that is sufficient for many years.

6 Key rollover

6.1 Periodic rollover

It is prudent to replace the keys regularly, not only to prevent exhaustive brute force attacks, but also so that the procedure of rolling keys does not grow stale.

DNSSEC Operational Practices (RFC4641 [9]), advises to replace KSKs once a year, and ZSKs once a month. There are, besides the cryptographic reasons, two reasons to set this to a year:

- This is a small enough timeframe for the procedure not to grow 'stale'
- This can be marked on a yearly calendar

However, as with key lengths, this recommendation is under discussion.

6.2 Emergency rollover

When a private key is lost, or compromised, a new public/private key set must be rolled out as soon as possible.

Recommendation (9): ***Have a documented procedure ready for the event where an emergency key rollover is necessary. This procedure should be tested on a regular basis, and updated if necessary.***

Keep in mind that automated rollover tracking systems (RFC5011 [12]) by validators may not be able to keep track of multiple emergency rollovers in quick succession. They typically need up to 30 days before a new (KSK) key is noticed and another 30 days before it is accepted (see RFC5011 [12]).

6.3 ZSK Rollover

Of the types of key to roll, Zone Signing Keys are the easiest, since no communication with external parties is necessary. If space and bandwith allows, one could just add the new key and signatures to the zone, wait for caches to be updated, and remove the old key and signatures. This scheme is called Double Signature Rollover in RFC4641 [9]. Note that when changing key algorithms, additional steps are necessary, see section 6.7.

Another scheme is the so-called Pre-publish Key Rollover. Here future keys are already added to the zone, but the signatures not yet. When the key is activated, caches should already have the new keys, and old signatures can be immediately removed. Because of the issue described in section 6.7, the pre-publish scheme can not be used to introduce a previously unused algorithm.

Both schemes are described in RFC4641 [9]. Be aware that there are errors in the provided figures. For clarity, we will include the correct tables on the next page. These tables are a direct copy from the errata to RFC4641, and explanation of the values in these tables can be found in that document.

Double signature ZSK rollover:

```
-----------------------------------------------------------------
initial              new DNSKEY          DNSKEY removal
-----------------------------------------------------------------
SOA0                 SOA1                SOA2
RRSIG10(SOA0)        RRSIG10(SOA1)       RRSIG11(SOA2)
                     RRSIG11(SOA1)

DNSKEY1              DNSKEY1             DNSKEY1
DNSKEY10             DNSKEY10            DNSKEY11
                     DNSKEY11
RRSIG1(DNSKEY)       RRSIG1(DNSKEY)      RRSIG1(DNSKEY)
RRSIG10(DNSKEY)      RRSIG10(DNSKEY)     RRSIG11(DNSKEY)
                     RRSIG11(DNSKEY)
-----------------------------------------------------------------
```

Pre-publish key rollover:

```
-----------------------------------------------------------------
initial          new DNSKEY       new RRSIGs       DNSKEY removal
-----------------------------------------------------------------
SOA0             SOA1             SOA2             SOA3
RRSIG10(SOA0)    RRSIG10(SOA1)    RRSIG11(SOA2)    RRSIG11(SOA3)

DNSKEY1          DNSKEY1          DNSKEY1          DNSKEY1
DNSKEY10         DNSKEY10         DNSKEY10         DNSKEY11
                 DNSKEY11         DNSKEY11
RRSIG1 (DNSKEY)  RRSIG1 (DNSKEY)  RRSIG1(DNSKEY)   RRSIG1 (DNSKEY)
RRSIG10(DNSKEY)  RRSIG10(DNSKEY)  RRSIG11(DNSKEY)  RRSIG11(DNSKEY)
-----------------------------------------------------------------
```

6.4 KSK Rollover as a child

For KSKs, additional steps are needed, because the keys need to be communicated to external parties.

If the parent zone of the one under administration has been signed, and offers secure delegations, the new keys must be communicated to the parent.

Depending on policies at the parent, or other operational agreements, either the DNSKEY RR or the DS RR needs to be present at the parent zone. Once the DS appears in queries to the parents, secure delegation can be verified with this key.

6.5 KSK Rollover as a trust anchor

If the parent is not signed, or the zone has other reasons to provide its DNSKEY RR as a trust anchor, then a secure channel needs to be used to publish the keys, and keep them updated. The usual method is to place them on a known and secured web site. This needs to be updated before the old signatures are removed, and if there is a notification mechanism, this to be secured, to protect against the attacks mentioned in section 2.3.

6.5.1 Automated rollover

While a number of automated key rollover approaches has been proposed, one has recently made it to publication in the form of RFC5011 [12]. This RFC provides a mechanism for automated trust anchor rollover. This RFC is fairly young, and there are not a lot of implementations yet. Both NLnet Labs and Sparta are known to be working on one.

6.6 Rollover of a child

When a child zone performs a KSK rollover, it needs to pass the public key or its DS record to the zone under administration to its parent.

This has the result that the parent at a delegation point must be able to handle more updates than before. Without DNSSEC, the only updates a parent gets are changes of the nameserver names or addresses. With DNSSEC, regular and emergency key rollovers must also be handled. In the latter case, update speed also becomes more important.

For every change, the relevant RRset needs to be signed again at the parent. Of course, the rest of the zone is unaffected, so it would be wise to have a system that can sign individual RRsets.

6.7 Algorithm Rollover

During the writing of this document, we have encountered a case that has to be handled differently; changing algorithms.

When having a succesful deployment of DNSSEC, it may eventually be necessary to change the cryptographic algorithms used to sign the zone data. For instance when a new attack has been found on the algorithm used.

Algorithm rollover will require additional steps that have, as of yet, not been documented publicly.

6.7.1 Algorithm downgrade protection

While not directly related to keys themselves, the problem originates in the following text from section 2.2 of RFC4035 [3]:

> There MUST be an RRSIG for each RRset using at least one DNSKEY of each algorithm in the zone apex DNSKEY RRset.

While this poses no problem when an admistrator rolls a key with an algorithm that is already present in the keyset, it can do so when introducing new or removing old algorithms.

Caches may have both the old keyset and the old list of signatures stored. When a new keyset is introduced, and the keyset happens to expire in the cache before the signatures do, the cache will retrieve the new keyset. Since it still has the old signatures, it will see no reason to update those.

Now the verifier will encounter a key with an algorithm for which there are no signatures. This is prohibited by the earlier statement in RFC4035, resulting in rejection of the data.

When removing an old algorithm, the same problem can occur when the signatures expire in the cache before the keyset.

6.7.2 Operational solution

A possible way for administrators to prevent this problem is to add a few steps to the rollover process.

- When adding a new algorithm type, add the new signatures, but omit the key at first. After all old data has expired from caches, add the new key.
- When removing an old algorithm type, remove the key first, but leave the signatures. After all old data has expired from caches, remove the old signatures.

6.7.3 Schematic representation of Operational solution

Here is a timeline of the original double signature rollover, based on the examples in RFC4641 [9], but simplified for the hypothetical case where there is only one key:

```
-------------------------------------------------------------
1: initial          2: new DNSKEY      3: DNSKEY removal
-------------------------------------------------------------
SOA0                SOA1               SOA2
RRSIG1(SOA0)        RRSIG1(SOA1)       RRSIG2(SOA2)
                    RRSIG2(SOA1)
```

```
DNSKEY1             DNSKEY1             DNSKEY2
                    DNSKEY2
RRSIG1(DNSKEY)      RRSIG1(DNSKEY)      RRSIG2(DNSKEY)
                    RRSIG2(DNSKEY)
-----------------------------------------------------------------
```

After step 2, there is the chance that in caches, the signature sets for the SOA, and other zone data, is not synchronized with the DNSKEY set. Which causes the problem mentioned earlier. The same problem occurs when the old algorithm is removed in step 3.

The timeline with the added steps is this:

```
-----------------------------------------------------------------
1 Initial           2 New RRSIGS        3 New DNSKEY
-----------------------------------------------------------------
SOA0                SOA1                SOA2
RRSIG1(SOA0)        RRSIG1(SOA1)        RRSIG1(SOA2)
                    RRSIG2(SOA1)        RRSIG2(SOA2)

DNSKEY1             DNSKEY1             DNSKEY1
RRSIG1(DNSKEY)      RRSIG1(DNSKEY)      DNSKEY2
                    RRSIG2(DNSKEY)      RRSIG1(DNSKEY)
                                        RRSIG2(DNSKEY)
-----------------------------------------------------------------
4 Remove DNSKEY     5 Remove RRSIGS
-----------------------------------------------------------------
SOA3                SOA4
RRSIG1(SOA3)        RRSIG2(SOA4)
RRSIG2(SOA3)

DNSKEY2             DNSKEY2
RRSIG1(DNSKEY)      RRSIG2(DNSKEY)
RRSIG2(DNSKEY)
-----------------------------------------------------------------
```

In step 2, the signatures for the new key are added, but the key itself is not. While in theory, the signatures of the keyset should always be synchronized with the keyset itself, it can be possible that RRSIGS are requested separately, so it might be prudent to also sign the DNSKEY set with the new signature.

After the cache data has expired, the new key can be added to the zone, as done in step 3.

The next step is to remove the old algorithm. This time the key needs to be removed first, before removing the signatures. The key is removed in step 4, and after the cache data has expired, the signatures can be removed in step 5.

The above steps ensure that during the rollover to a new algorithm, the cryptographic integrity of the zone is never compromised.

7 Private key storage

Due to the importance of the private keys, their storage should be safe. This is especially true for KSKs and trust anchors.

Because of the design of DNSSEC, these keys need not be continually available on authoritative servers; ZSKs are needed when a zone is signed, when zone data is changed or when zone data signatures expire. KSKs are only needed when a ZSK is rolled, or the signatures on the keyset expire, so that the DNSKEY RRset can be signed again.

When choosing a storage medium for private keys, a trade-off must be made on several points, most of these against cost. Since the keys need to be used for signing, we shall not only take the medium itself into account, but also the means to get the private keys to the signing system.

Cost does not only consist of the purchase and maintenance prices of hardware and software. One should also consider the amount of effort needed should a key be lost or compromised. For instance, an emergency rollover for a ZSK costs a lot less than one of a trust anchor. Therefore, we advise that these choices are made separately for KSKs and ZSKs.

First of all, one should look at the risk level. For a so-called 'vanity-zone', the risk level would probably be very low. A 'high' zone, such as a TLD, or the root zone, and the root zone itself, would have a high risk, since the impact of a succesful attack would be much, much greater. A zone containing important information, like that of a banking website, would also be on the higher end.

A choice to make is how the machine that signs the data has access to the private keys. This could be direct access (online) or no access without some form of physical action, for instance by use of some smart card (offline). Or it could reside on a machine that is only available through a well-defined and possibly secured channel. For instance on a separate machine that is only accessible through a console interface. This is also referred to as a shielded system.

When using an online mechanism, the keys could reside on an HSM [1], with its own electronic and physical protection. FIPS-140 [10] defines a number of levels of security, that a certain module can be certified for, in short:

- Level 1: This is the lowest level. A security level 1 cryptographic module does not have to have physical protection, and only needs to incorporate one approved algorithm or function.
- Level 2: Security level 2 requires tamper evidence to be added to the module, as well as role-based authentication.
- Level 3: In addition to tamper evidence, for security level 3, a module must also provide tamper resistance. This level also requires identity-based authentication.
- Level 4: This is the highest level specified by FIPS 140. This level requires complete protection around the cryptographic module, detecting and re-

[1] See section 9.3 for more information on software with HSM support.

sponding to all unauthorized attempts at physical access, as well as environmental anomalies (power fluctuations, extreme temperatures).

For offline storage, an option is to store the key on a removable medium, so that it can be placed in a safe location. One could use a medium that has its own access control. If the medium should be stolen, the attacker would still need a password to use the key data.

Below is a table where we show a possible system for the above choices, roughly in descending cost. These should not be seen as hard recommendations; some solutions will merge certain choices, and others might split up choices in several smaller ones. For instance, a DNSSEC appliance could have a high FIPS level, but still need to be offline.

When using an offline system, we also mention 'reviewed procedures'. These will of course be necessary in any design. We mention them here because there is a procedural number of actions that must be taken for the system to do its work, and it should be well-defined who has access to the keys and in which manner.

Risk	On-/offline	Advised system
High Risk	Online	FIPS Level 4 HSM
High Risk	Offline	Reviewed Procedures, Physical Safe [a]
Medium Risk	Online	FIPS-2 HSM, or shielded normal system [b]
Medium Risk	Offline	Reviewed Procedures
Low Risk	Online	Normal connected or local system
Low Risk	Offline	

[a] The medium itself can also have extra access-control

[b] A shielded system is a system that is connected, but only accessible through a well-defined single channel

8 Public key publication

Whether for original configuration of verifying resolvers, or for updating rolled keys, a secure mechanism must be used to publish public keys. Assuming that a KSK/ZSK mechanism is chosen, we will only handle KSKs here. If the zone is only signed with ZSKs, these should be treated as KSKs for this chapter.

8.1 Through the parent

If the parent zone is signed, it should have a mechanism to provide new and updated keys. Depending on their procedures, these should be passed as either the DNSKEY Resource Record, or in the form of a DS Resource Record.

8.2 DLV

ISC operates the so-called DNSSEC Look-aside Validation registry. This is a central repository where public keys can be stored and published for automatic use in DNSSEC verifiers.

From their website:

```
DLV (DNSSEC Look-aside Validation) is a non-IETF extension to the
DNSSECbis protocol extensions that allow the early deployment of
DNSSECbis in the absence of a signed DNS tree at the root, Top Level
Domain and near-top levels.

DLV provides an additional entry point from which to obtain DNSSEC
validation information. In the absence of a signed root zone, users
wanting to make use of DNSSECbis would need to maintain a series of
trusted keys (anchors) in their name server configurations,
corresponding to the domains that publish the cryptographic keys they
use to sign their zones.

Maintaining all this information up to date can quickly turn into an
unmanageable task. DLV allows the user to configure a single point of
entry, from which, via DLV Resource Records and dns lookups, the DNS
tree can be searched looking for applicable Delegation signers.
```

At the moment, the only validator implementation that fully supports DLV is ISC's BIND. DLV support in NLnet Labs' Unbound is worked on, and will probably be available in the next release.

8.3 Out of band

With the current adoption rate of DNSSEC, it will benefit users to have an external way to find and configure their secure entry points. Therefore it is, at least until the root zone is signed, advisable for important zones to publish their public keys separately. At the moment, the most commonly used way for this is

an SSL-secured web site. Another option could be a PGP-secured mailing list for updates.

Such a mechanism should have at least two features:

- Since keys are published, this mechanism is an obvious point of attack, and therefore needs to be protected for both integrity and authenticity. Since the objects distributed through this system are public keys, confidentiality is less of an issue.

- There should be a notification system, so users who have configured their keys manually can know when to update them. If their keys should go stale, all data from the zones of these keys will be marked as bogus.

8.4 In-band

Apart from publication through a signed parent zone, there is no in-band mechanism for initial publication of new keys. However, for key rollover, the mechanism from RFC5011 [12] can be used.

There are tools that simply fetch the DNSKEY list at configured trust anchors, and if there is new data that can be verified with the old anchors, the anchors are updated. [2]

8.5 Keys as secure entry points

Administrators should always assume that their keys are used as secure entry points, and should treat them as such. Even if DLV is used, the parent zone is signed and updated, users can still configure the keys as secure entry points.

Therefore we recommend that either an out-of-band publication system is used in conjuction with the other publication mechanism, or a very specific statement is issued through such a channel that the keys should not or no longer be used as secure entry points, and that doing so, and the results thereof, are entirely the responsibility of the users configuring them as such.

Recommendation (10): ***Either an out-of-band publication system should be used in conjuction with the other publication mechanism, or a very specific statement should be issued through such a channel that the keys should not or no longer be used as secure entry points, and that doing so, and the results thereof, are entirely the responsibility of the users configuring them as such.***

8.6 Key updates of children

When a zone contains delegations to zones under the administration of other parties, those parties would need to have a secure channel to update the DS value at their parent (this zone).

[2] Unbound has such a script in the contrib/ directory called update-anchor.sh. A similar tool exists for BIND.

If only the DS records are stored, and the policy for the DS hashing algorithm is ever changed, the zone administrators would have to ask every single child zone for new DS records. If the parent stores the DNSKEY records of their children, it could derive the DS records by itself.

Recommendation (11): ***For zones with a lot of children, such as a TLD, is would be smart no to store just the DS record, but also the DNSKEY record from which it was derived.***

9 Management systems and tools

There are several projects underway to create management systems and tools for DNSSEC.

Here is a list, it is not meant to be exhaustive, but merely to give the reader of this document an idea of what is available, and where current focus lies.

9.1 Specific Key Management Software

- RIPE NCC has a toolset for key and zone management based on Net::DNS. It provides a key database, command-line tools and interfaces to that database, and a SOAP client for signers. Information, documentation and downloads are available at

  https://www.ripe.net/projects/disi/dnssec_maint_tool/

- NLnet Labs is working on a client implementation for RFC5011 [12] called Autotrust.

9.2 DNSSEC Libraries and toolsets

- SPARTA Inc. created and maintains a DNSSEC Toolkit called DNSSEC-Tools. Among more general DNSSEC tools for both users and zone administrators, it also provides an application and daemon for key rollover.

  http://www.dnssec-tools.org/

- Verisign provides a collection of Java-based tools for DNSSEC, based on DNSjava.

  http://www.verisignlabs.com/dnssec-tools/

- Net::DNS::SEC is an extension to Net::DNS. It is a perl module for DNSSEC applications.

  http://www.net-dns.org/

9.3 HSM Support in Software

Several patches and tools provide support for HSM in software.

- .SE has a utility that support PKCS #15 smartcard storage and conversion to DNSSEC records.

  http://opensource.iis.se/trac/dnssec/browser/pkcs15-dnssec

- Richard Lamb has created patches for BIND that implement PKCS #11.

  http://www.xtcn.com/ lamb/pkcs11HSMtools.tar.gz

- ldns from NLnet Labs has native PKCS #11 support through OpenSSL.

  http://www.nlnetlabs.nl/projects/ldns/

9.4 General Software Information

For more information, consult the DNSSEC website at
http://www.dnssec.net
or the DNSSEC-deployment software tracker at
http://www.dnssec-deployment.org/tracker/

About NLnet Labs

NLnet Labs was founded in 1999 by Stichting NLnet to develop, implement, evaluate and promote new protocols and applications for the Internet.

The goal of NLnet Labs is to contribute knowledge to the Internet. This can be achieved by software development, and also by educating people to develop software or deploy protocols. NLnet Labs' staff therefore not only focuses on software development defined in projects, but also on collaboration with other organizations. The budget of NLnet Labs is based on long term investment for development with a staff of five to six people.

About the sponsor of this paper

.SE (The Internet Infrastructure Foundation), founded as a non-profit organisation in 1997, is responsible for the top-level Internet domain for Sweden, .se. .SE's core operations are registration of domain names and the administration and technical operation of the national domain name register.

Within the framework of these operations, .SE works to ensure the positive development of the Internet in Sweden over the long term, for the benefit of users, operators, businesses, authorities, universities and others. This way, .SE wants users of domain-name services to have access to high-quality, robust services on reasonable terms.

Legal Notice

References

[1] P. Antonov and V. Antonova. *Development of the attack against RSA with low public exponent and related messages*. In *CompSysTech '07: Proceedings of the 2007 international conference on Computer systems and technologies*, pages 1–8, New York, NY, USA, 2007. ACM.

[2] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. *DNS Security Introduction and Requirements*. RFC 4033 (Proposed Standard), March 2005. `http://www.ietf.org/rfc/rfc4033.txt`.

[3] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. *Protocol Modifications for the DNS Security Extensions*. RFC 4035 (Proposed Standard), March 2005. `http://www.ietf.org/rfc/rfc4035.txt`, (Updated by RFC 4470).

[4] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. *Resource Records for the DNS Security Extensions*. RFC 4034 (Proposed Standard), March 2005. `http://www.ietf.org/rfc/rfc4034.txt`, (Updated by RFC 4470).

[5] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. *Recommendation for Key Management*, March 2007. `http://csrc.nist.gov/publications/pubssps.html#800-57`.

[6] Daniel Bleichenbacher. *Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1*. In *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, pages 1–12, London, UK, 1998. Springer-Verlag.

[7] D. Eastlake 3rd. *DSA KEYs and SIGs in the Domain Name System (DNS)*. RFC 2536 (Proposed Standard), March 1999. `http://www.ietf.org/rfc/rfc2536.txt`.

[8] J. Jansen. *Use of SHA-2 algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC*, July 2008. `http://tools.ietf.org/wg/dnsext/draft-ietf-dnsext-dnssec-rsasha256/`, (Work in progress).

[9] O. Kolkman and R. Gieben. *DNSSEC Operational Practices*. RFC 4641 (Proposed Standard), September 2006. `http://www.ietf.org/rfc/rfc4641.txt`.

[10] NIST. *Security Requirements for Cryptographic Modules*, May 2005. `http://csrc.nist.gov/publications/pubsfips.html`.

[11] Richard C. Schroeppel and Donald Eastlake 3rd. *Elliptic Curve Keys and Signatures in the Domain Name System (DNS)*, March 2007. `http://tools.ietf.org/wg/dnsext/draft-ietf-dnsext-ecc-key/`, (Work in progress, Expired).

[12] M. StJohns. *Automated Updates of DNS Security (DNSSEC) Trust Anchors.* RFC 5011 (Proposed Standard), September 2007. `http://www.ietf.org/rfc/rfc5011.txt`.

[13] Debian Security Team. *Debian Security Advisory: OpenSSL – Predictable random number generator*, May 2008. `http://www.debian.org/security/2008/dsa-1571`.