# DNSSEC Key State Transitions

Yuri Schaeffer, Yuri@NLnetLabs.nl

July 1, 2011

## 1 Three Laws of DNSSEC

**First Law** An Enforcer may not injure a zone by revoking its DS record.

**Second Law** An Enforcer must keep a secure KSK or an insecure KSK if that does not conflict with he first rule.

**Third Law** An Enforcer must protect its ZSK security unless revoking does not conflict with the first or second rule.

**Zeroth Law** An enforcer may not harm a zone, or, by inaction, allow zones to come to harm.

> “ Trevize frowned. “How do you decide what is injurious, or not injurious, to humanity as a whole?”
>
> “Precisely, sir,” said Daneel. “In theory, the Zeroth Law was the answer to our problems. In practice, we could never decide. A human being is a concrete object. Injury to a person can be estimated and judged. Humanity is an abstraction.” ”
>
> –Foundation and Earth

## 2 Warning

This document is a rough explanation of an idea. It is based on earlier work and may use but not introduce terminology of an other document. Possibly introduce completely new notations, skip reasoning steps without warning (well, you are warned now) and be plain wrong. This document is considered unsuitable for young children and Liberal-Arts majors. Read at OWN RISK.

# Contents

**1 Three Laws of DNSSEC** **1**

**2 Warning** **1**

**3 Introduction** **2**

**4 Cache Centered Approach** **3**

**5 Keys** **3**
- 5.1 Keyring . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
- 5.2 Key States . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
- 5.3 Goal . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
- 5.4 Role . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

**6 Model** **5**
- 6.1 Notation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
- 6.2 Rules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
- 6.3 Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

**7 Rules explained** **8**

**8 Examples** **9**
- 8.1 ZSK Double Signature . . . . . . . . . . . . . . . . . . . . . 9
- 8.2 Other examples . . . . . . . . . . . . . . . . . . . . . . . . . 10

## 3 Introduction

A key roll-over is a process that runs over time, at any time each key has a state. Matthijs Mekking introduced the idea[1] that a key has a private part, a public part, and a parent part. Each of these parts have their own state-machine an together represent a 'key-state'.

This idea forms the basis for this document and allows us to generalize much of the logic. All parts of the key-state now represent records which are actually published to the world. The state of a record tells if and how well the record is known to caching resolvers[1]. As a result each record now has exactly the same state machine. Furthermore instead of 3 parts we use 4 parts per key.

With these ingredients we can formalize the boundries of DNSSEC. We say a zone is in a valid DNSSEC state if a validator can build a chain of trust from that state. Alternatively, a unsigned zone is also considered valid, as long as it is not bogus[2]. In terms of end-user, if the user is able to get an address for a name, the zone is valid. With our formal definition we can test the current state for validity and have a precise way of deciding we can take a transition to a next key state.

[1]This document mixes the terms cache, resolver and validator. In all cases a DNSSEC validating caching resolver is meant.

[2]Bogus: The validating resolver has a trust anchor and a secure delegation indicating that subsidiary data is signed, but the response fails to validate for some reason: missing signatures, expired signatures, signatures with unsupported algorithms, data missing that the relevant NSEC RR says should be present, and so forth. (RFC4033)

## 4 Cache Centered Approach

In a cache centered approach we do not define how rollovers should work exactly. The state of a key is represents how well it is know to *any* caching validator. Either all caches have a record, no caches have a record, or some caches *might* have it en some caches *might not*. In the latter case we must simply wait long enough and the record will enter one of the certain states again. We will discuss the key states in section 5.2.

Each key has a goal, the goal is either to be published and to be known to all caches around the world or to none at all. A system can make any state transition as long as it makes sure that the validity of a zone in general is not compromised. This does also imply that a key's goal can be changed at any time without the zone going bogus. E.g. if a key has a desire to disappear but is the only key left it will stay on duty for as long as necessary. Also, new keys can be introduced at any time.

This approach has a number of advantages. Here, in contrast to a roll-over centered approach, keys have no direct relation to each other. The system does not try to roll from one specific key to another specific key but rather satisfy all goals while remaining valid as a whole. Essentially the roll-over is a side effect of the strive to satisfy key goals. New keys can be introduced and goals can be redefined at any time without a problem. This makes the system agile, robust, and capable of handling unexpected situations. Another advantage is that our system will 'think' the same was as the validators we are trying to satisfy. It always knows how resolvers (might) see the data and makes keeping that data valid its core business.

## 5 Keys

### 5.1 Keyring

Although zones can share key material the records are published independently. As a result each zone must have its own collection of key states. Let us refer to this collection as a keyring denoted by $\mathbb{K}$. A Keyring and a Zone have a 1-on-1 relation.

Keys not known to any resolver can freely be added to or removed from the keyring. Similar, if a key is in none of the keyrings the key material can be purged safely from the keystorage.

### 5.2 Key States

The state of a key is publicly represented by the DS record, DNSKEY record, RRSIG DNSKEY record and the RRSIGs over all the data in the zone. The latter consists of multiple records but all share the same state, although technically incorrect we will refer to it as the RRSIG record.

Because each of the four records can be introduced at a different time and at a different pace, all need their own state machine (Figure 1).

**Reputation** The state of a resource record is defined by its reputation in world's caches. There are 2 certain states, *Hidden* and *Omnipresent*. In the

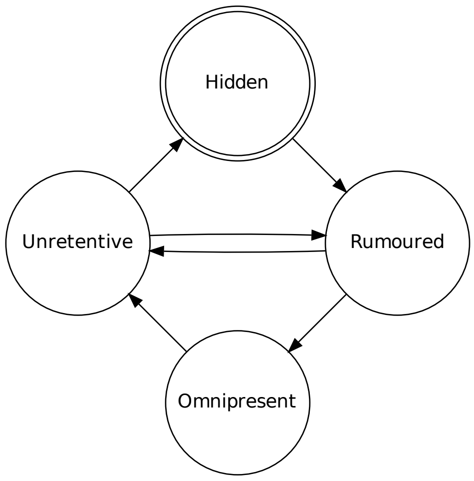

Figure 1: State diagram for individual records.

first state a resource record is not available in any cache (any longer or at start). Similarly in the latter state we are sure all caches have a copy of the record or can at least obtain it (i.e. they have no old resource record set with expired TTL).

The two other states represent uncertainty: *Rumoured* and *Unretentive.* Both mean that some caches do and some don't know about the resource record. In the former state a record is being actively published and will somewhere in the future be known by all caches. While the latter state the record is not published (any longer) and will disappear from collective memory over time.

Since DNS involves caches and TTLs, it is entirely possible for a record being in the *Rumoured* state while in fact every cache in the world holds the record. The problem is the uncertainty, we have no way to know a record is fully propagated other than waiting the worst case propagation time. This is also true for the *Unretentive* state.

**Summary** A record is always in one of these four states:

| | |
|---|---|
| *Hidden*: | The record is in no cache at all. |
| *Rumoured*: | The record is published but not every cache might be aware. |
| *Omnipresent*: | Every cache has this record. |
| *Unretentive*: | The record is withdrawn but some caches might still have it. |

**Formal definition of record states** Let $r$ be a record, *Announced* the collection of records that are actively published, and $\mathbb{C}$ the collection of all caches in the world. The states are defined by:

$$\begin{array}{lcl}
r \in \mathit{Hidden} & \equiv & \forall c \in \mathbb{C} \cdot r \notin c \\
r \in \mathit{Rumoured} & \equiv & \exists c \in \mathbb{C} \cdot r \notin c \land \exists c \in \mathbb{C} \cdot r \in c \land r \in \mathit{Announced} \\
r \in \mathit{Omnipresent} & \equiv & \forall c \in \mathbb{C} \cdot r \in c \\
r \in \mathit{Unretentive} & \equiv & \exists c \in \mathbb{C} \cdot r \notin c \land \exists c \in \mathbb{C} \cdot r \in c \land r \notin \mathit{Announced}
\end{array}$$

**Ordering** We also define a ordering on record states so we can compare states. *hidden* < *rumoured* < *omnipresent* < *unretentive*. This represents the normal lifecycle for a record as depicted in Figure 1.

## 5.3 Goal

Each key in the keyring has a goal. The goal is an internal drive for the individual records to reach a certain state. The goal is either 'well-known' or 'not-known'. While the goal is 'well-known' (the key is said to be *introducing*) the record tries to go to the Omnipresent state. In the other case its direction of movement is towards the hidden state and is said to be *outroducing*. When a record has fulfilled its goal the record is said to be stable. The record can become unstable again only if the goal changes (or the state changes somehow by user intervention, but this is potentially harmful).

## 5.4 Role

A key has one or two roles. It is a 'key Signing key' (KSK), a 'zone signing key' (ZSK), or both — combined signing key (CSK). This document does not spend too much time on these roles as our model is not concerned with the difference. It simply checks whether or not there is a key with certain properties. For example, a KSK has no RRSIG record. Every statement about its RRSIG record evaluates to False. A ZSK has no DS and RRSIG DNSKEY record. A CSK has all records.

# 6 Model

## 6.1 Notation

For readability let $D$ denote a DS record, $K$ the DNSKEY, $k$ the RRSIG DNSKEY, and $S$ the RRSIG. The state of a record is indicated with superscript and can be found in Table 1. These states correspondent directly with the states from Figure 1. I.e. *hidden* $\equiv -$, *rumoured* $\equiv\uparrow$, *omnipresent* $\equiv +$, *unretentive* $\equiv\downarrow$. The subscript of a record denotes the key it belongs to. The current evaluated key has subscript $i$. E.g: the statement "the DS of key $z$ is in the Omnipresent state" can be written as $D_z^+$.

| | | P | |
|---|---|---|---|
| | | $\leq 1$ | 1 |
| direction | in | $\uparrow$ | $+$ |
| | out | $\downarrow$ | - |

Table 1: Notation of states in superscript

## 6.2 Rules

The system uses three rules, Equation (1)(2)(3). To see if we can transition a record to the next state we first test the three rules. They *should* all be True, if not we apparently have an invalid zone on our hands. This isn't our fault (presumably[3]) but we have to deal with it.

Next we evaluate the rules again but pretend we gave the record the requested state. At least the rules with a positive outcome in the first test *must* still be positive if we were to apply this state transition. Otherwise we may not make the transition as it would make the zone *less valid.*

Allowing rules to be negative in the first test while prohibit positive rules become negative prevents locks in our statemachine and will to some extend repair invalid situations in a graceful manner. Also, it allows unsigned zones to exist without special tricks.

The three rules *should* be true for each record of all keys. The rules will be explained in a later section.

$$rule1(i) \quad : \quad \exists x \cdot D_x^{\uparrow} \vee D_x^{+} \tag{1}$$

$$\begin{aligned}
rule2(i) \quad &: \quad \forall x \cdot (D_x^{-} \vee \exists y \cdot K_y^{+} k_y^{+} (D_x = D_y)) \\
&\vee \quad \exists x \cdot D_x^{+} K_x^{+} k_x^{+} \\
&\vee \quad \exists x, y \cdot D_x^{\uparrow} K_x^{+} k_x^{+} D_y^{\downarrow} K_y^{+} k_y^{+} \\
&\vee \quad \exists x, y \cdot D_x^{+} K_x^{\uparrow} k_x^{\uparrow} D_y^{+} K_y^{\downarrow} \\
&; \quad alg(i) = alg(x) = alg(y)
\end{aligned} \tag{2}$$

$$\begin{aligned}
rule3(i) \quad &: \quad \forall x \cdot (K_x^{-} \vee \exists y \cdot S_y^{+} (K_x = K_y)) \\
&\vee \quad \exists x \cdot K_x^{+} S_x^{+} \\
&\vee \quad \exists x, y \cdot K_x^{\uparrow} S_x^{+} K_y^{\downarrow} S_y^{+} \\
&\vee \quad \exists x, y \cdot K_x^{+} S_x^{\uparrow} K_y^{+} S_y^{\downarrow} \\
&; \quad alg(i) = alg(x) = alg(y)
\end{aligned} \tag{3}$$

## 6.3 Algorithm

The defined rules are just for deciding if a single record may change its state to the next. Algorithm 1 describes how this can be done for a whole zone while also take timing into account.

Three functions might need more explanation:

**desiredState(state, goal)** returns the state a record wants to change to given its movement direction and current state.

**transitionAllowed(keyring, key, record, state)** Evaluates the rules and checks if *record* can be brought to *state* with respect to the DNSSEC validity.

[3]The system should never bring the zone to an invalid state, if it does it is considered a serious bug!

**Algorithm 1** Advance keys within a zone, return time for next run.

```
$nextRun \leftarrow \infty$
repeat
  $change \leftarrow \bot$
  for all $key \in keyring$ do
    for all $record \in key$ do
      $nextState \leftarrow desiredState(state(record), goal(key))$
      if $nextState = state(record)$ then
        {This record is in a stable state}
        continue
      end if
      if not $transitionAllowed(keyring, key, record, nextState)$ then
        {This transition would make the zone invalid}
        continue
      end if
      $t \leftarrow transitionTime(type(record), state(record), nextState, lastChange(record))$
      if $t \geq now()$ then
        {We are not allowed to make the transition at this time}
        $nextRun \leftarrow minimum(t, nextRun)$
        continue
      end if
      $state(record) \leftarrow nextState$
      $lastChange(record) \leftarrow now()$
      $change \leftarrow \top$
    end for
  end for
until not $change$
return $nextRun$
```

**transitionTime(type, state, state, time)** The absolute time after which the transition between the two states can take place given the type of record and the time of last change. This is policy specific.

## 7 Rules explained

In a nutshell, the first rule makes sure the zone is kept signed at all times, the second keeps the KSK in a valid state and the third the ZSK. The three rules do influence each other. For example rule 3 does allow having no ZSK, but only if there is no KSK. Rule 2 does allow having no KSK but only if there is no DS published. Rule 1 seems simple but it is the only thing that keeps the zone from going unsigned.

Disabling rule 1 will allow both other rules to go to unsigned (in a gracefull manner).

$$rule1(i) \quad : \quad \exists x \cdot D_x^{\uparrow} \vee D_x^{+} \tag{4}$$

At all times there must be a DS published (4). The only exception is when the user wants his zone unsigned, this must be explicitly stated. If so, this rule must skipped. Skipping in other situations is possible, but ill advised. The enforcer may do a rollover via an unsigned zone.

In words: there must always exist a DS record introducing or omnipresent. It does not matter what algorithm or which key this DS belongs to. This seems incomplete but really forms a fundament for rule number 2.

$$\begin{aligned}
rule2(i) : &\quad \forall x \cdot (D_x^{-} \vee \exists y \cdot K_y^{+} k_y^{+} (D_x = D_y)) && \text{(5a)}\\
\vee &\quad \exists x \cdot D_x^{+} K_x^{+} k_x^{+} && \text{(5b)}\\
\vee &\quad \exists x, y \cdot D_x^{\uparrow} K_x^{+} k_x^{+} D_y^{\downarrow} K_y^{+} k_y^{+} && \text{(5c)}\\
\vee &\quad \exists x, y \cdot D_x^{+} K_x^{\uparrow} k_x^{\uparrow} D_y^{+} K_y^{\downarrow} && \text{(5d)}\\
; &\quad alg(i) = alg(x) = alg(y) && \text{(5e)}
\end{aligned}$$

Rule 2 describes the dependency of the DS record on the DNSKEY and its RRSIG. The subequations represent the different scenarios that are valid, at least one of these scenarios must be true at all times. The only keys that need to be considered are the keys with the same algorithm as the input key $i$ (5e).

Subequation (5b) is the trivial case: If there is a key with the DS, DNSKEY and RRSIG DNSKEY fully propagated all other keys of this algorithm can be in any state. No further requirements.

Similar subequation (5c) requires two keys swapping DS records. Again, if this is the case, the zone is valid for the KSK and other keys can be in any state without consequences.

Subequation (5d) also requires two keys, but now they are swapping DNSKEYs. Notice the outroduction of the signature is omitted in this subequation ($k_y^{\downarrow}$), this is not an error. Including it would block the DNSKEY $k_y$.

Finally the unsigned case, subequation (5a). It applies to zones unsigned *for this algorithm*, which includes unsigned for *any* algorithm. In the unsigned/partially signed case no DS may be published. If however a DS of key $x$

is published anyway there must be a key $y$ with the DNSKEY set propagated and the DS in the same state as key $x$. Like everywhere else $x$ and $y$ may be the same key.

Let us review an example (ignoring rule 1 and 2). Say we have two keys: $D_x^+ K_x^+ k_x^+$ and $D_y^+ K_y^- k_y^-$. We want the zone to become unsigned. At time 0 the zone is valid because of (5b) and we can therefore outroduce the DS of $y$. Now, because of (5a) me may do the same with the DS of $x$ at the same time. When both are hidden the last mentioned rule allows the remaining DNSKEY and RRSIG DNSKEY to do whatever they want.

$$
\begin{array}{rrr}
rule3(i): & \forall x \cdot (K_x^- \vee \exists y \cdot S_y^+ (K_x = K_y)) & \text{(6a)} \\
\vee & \exists x \cdot K_x^+ S_x^+ & \text{(6b)} \\
\vee & \exists x, y \cdot K_x^{\uparrow} S_x^+ K_y^{\downarrow} S_y^+ & \text{(6c)} \\
\vee & \exists x, y \cdot K_x^+ S_x^{\uparrow} K_y^+ S_y^{\downarrow} & \text{(6d)} \\
; & alg(i) = alg(x) = alg(y) & \text{(6e)}
\end{array}
$$

The third rule works exactly the same as rule 2 but reasons about the ZSK part.

## 8 Examples

Names for rollovers are taken from "DNSSEC Key Timing Considerations Follow-Up". The tables in the following sections contain different kind of roll overs. Each row in the table represents a time step and the first row is the start situation. When a key does not change during a time step the record cells are left blank.

### 8.1 ZSK Double Signature

Without any policy directives a double signature rollover will be executed as this is the fastest way.

| KSK1 (in) | | | ZSK1 (out) | | ZSK2 (in) | | Time |
|---|---|---|---|---|---|---|---|
| $A^+$ | $B^+$ | $C^+$ | $B^+$ | $D^+$ | $B^-$ | $D^-$ | |
| | | | | | $B^{\uparrow}$ | $D^{\uparrow}$ | 0 |
| | | | $B^{\downarrow}$ | $D^{\downarrow}$ | $B^+$ | $D^+$ | $MaxTTL(key, sig)$ |
| | | | $B^-$ | $D^-$ | | | $MinTTL(key, sig)$ |
| Total time | | | | | | | $TTL(sig) + TTL(key)$[1] |

[1] The total time does not seem obvious from this table. The 'fast' record set will have to wait while the 'slow' record set is swapping keys.

Table 2: ZSK Double Signature rollover

Let us walk trough the example in Table 2. We'll evaluate each record from left to right. The KSK is ignored in our description as there are no changes during the rollover.

Since ZSK1 has is outroducing its records try to move to the unretentive state ($\downarrow$). This may not happen yet. For example the desired state for the DNSKEY (B) of ZSK1 is unretentive. Currently using ZSK1 as key $i$ all four rules are true. If we would now use the new state rule2 would become false. the $A_x^{\uparrow} B_x^{+} C_x^{+}$ part would no longer be true. Therefore we may not take this transition at this moment and we may stop look at the other rules, although rule3 would also become false.

For both the DNSKEY as the RRSIG of ZSK2 all rules are true prior to a transition. After a transition they would still be true, so we can take these records safely to the nest state.

In the second time step (row 3) we can still not progress ZKS1 initially. But, given enough time passed we may bring ZSK2's records to the next state. Suppose both the DNSKEY and RRSIG of ZSK2 change at the same time. In this very same time step we are now suddenly allowed to outroduce the records of ZSK1. Because we keep looping over all records of all keys till no record changes anymore the order of evaluation does not matter for the effectivity of the algorithm (it might affect the efficiency however).

The last time step has no actions for ZSK2 as all its records are in a stable state. The records in ZSK1 can go to hidden ($-$). Their state has no effect on any of the rules as ZSK2 causes all rules to be true anyway.

## 8.2 Other examples

Further examples are not explained in detail and are here for reference.

| KSK1 (in) | | | ZSK1 (out) | | ZSK2 (in) | | Time |
|---|---|---|---|---|---|---|---|
| $A^+$ | $B^+$ | $C^+$ | $B^+$ | $D^+$ | $B^-$ | $D^-$ | |
| | | | | | $B^{\uparrow}$ | $D^-$ | 0 |
| | | | $B^+$ | $D^{\downarrow}$ | $B^+$ | $D^{\uparrow}$ | $TTL(key)$ |
| | | | $B^{\downarrow}$ | $D^-$ | $B^+$ | $D^+$ | $TTL(sig)$ |
| | | | $B^-$ | $D^-$ | | | $TTL(key)$ |
| Total time | | | | | | | $2 \times TTL(key) + TTL(sig)$ |

Table 3: ZSK Pre-Pubplucation rollover

| KSK1 (in) | | | ZSK1 (out) | | ZSK2 (in) | | Time |
|---|---|---|---|---|---|---|---|
| $A^+$ | $B^+$ | $C^+$ | $B^+$ | $D^+$ | $B^-$ | $D^-$ | |
| | | | | | $B^-$ | $D^{\uparrow}$ | 0 |
| | | | $B^{\downarrow}$ | $D^+$ | $B^{\uparrow}$ | $D^+$ | $TTL(sig)$ |
| | | | $B^-$ | $D^{\downarrow}$ | $B^+$ | $D^+$ | $TTL(key)$ |
| | | | $B^-$ | $D^-$ | | | $TTL(sig)$ |
| Total time | | | | | | | $2 \times TTL(sig) + TTL(key)$ |

Table 4: ZSK Double RRSIG rollover

| KSK1 (out) | | | ZSK1 (out) | | CSK1 (in) | | | | Time |
|---|---|---|---|---|---|---|---|---|---|
| $A^+$ | $B^+$ | $C^+$ | $B^+$ | $D^+$ | $A^-$ | $B^-$ | $C^-$ | $D^-$ | |
| | | | | | $A^-$ | $B^-$ | $C^-$ | $D^\uparrow$ | 0 |
| | | | | | $A^-$ | $B^\uparrow$ | $C^\uparrow$ | $D^+$ | TTL(sig) |
| $A^\downarrow$ | $B^+$ | $C^+$ | | | $A^\uparrow$ | $B^+$ | $C^+$ | $D^+$ | TTL(key) |
| $A^-$ | $B^\downarrow$ | $C^\downarrow$ | $B^\downarrow$ | $D^+$ | $A^+$ | $B^+$ | $C^+$ | $D^+$ | TTL(ds) |
| $A^-$ | $B^-$ | $C^-$ | $B^-$ | $D^\downarrow$ | | | | | TTL(key) |
| | | | $B^-$ | $D^-$ | | | | | TTL(sig) |
| Total time | | | | | | | | | $2 \times TTL(key) + 2 \times TTL(sig) + TTL(ds)$ |

Table 5: Split Key to Single Key Algorithm rollover

## References

[1] Matthijs Mekking, *DNSSEC Key Timing Considerations Follow-Up.* Internet-Draft, draft-mekking-dnsop-dnssec-key-timing-bis-00, February 25, 2011.