DNSSEC Key State Transitions

Yuri Schaeffer, Yuri@NLnetLabs.nl

June 21, 2011

1 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.

1.1 Incompleteness / TODO

Minimize Flags define how the (in this document yet undefined) minimize flags should exactly modify the rules.

Notation Brush up notation of rules.

Contents

2 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.

3 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 4.2.

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.

[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 Keys

4.1 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.

4.2 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).

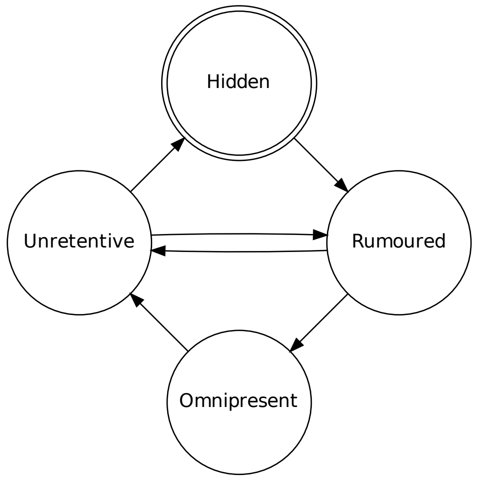

Figure 1: State diagram for individual records.

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 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. $\mathit{hidden} < \mathit{rumoured} < \mathit{omnipresent} < \mathit{unretentive}$. This represents the normal lifecycle for a record as depicted in Figure 1.

4.3 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).

4.4 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.

5 Model

5.1 Notation

For readability let A denote a DS record, B the DNSKEY, C the RRSIG DNSKEY, and D 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 A_z^+.

		P	
		≤ 1	1
direction	in	$\uparrow$	$+$
	out	$\downarrow$	-

Table 1: Notation of states in superscript

Furthermore, the certain states are a subset of the uncertain states. So a Hidden record is also said to be Unretentive. $A_x^- \rightarrow A_x^\downarrow$.

5.2 Rules

The system uses four rules, Equation (1)(2)(3)(4). 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 four rules *should* be true for each record of all keys. The rules will be explained in a later section.

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

$$\begin{aligned}
rule1: &\quad A_x^{\uparrow} && (1)\\
rule2: &\quad A_x^{\neg\uparrow}(A_i^{-} \vee B_i^{+}C_i^{+}) \vee A_x^{\uparrow}B_x^{+}C_x^{+} \vee A_x^{+}B_x^{\uparrow}C_x^{\uparrow}A_y^{+}B_y^{\downarrow}C_y^{\downarrow} && (2)\\
rule3: &\quad B_x^{\neg\uparrow}(B_i^{-} \vee D_i^{+}) \vee B_x^{\uparrow}D_x^{+} \vee B_x^{+}D_x^{\uparrow}B_y^{+}D_y^{\downarrow} && (3)\\
rule4: &\quad B_i \geq C_i && (4)
\end{aligned}$$

5.3 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.

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 $nextRun$
```

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.

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.

6 Rules explained

$$A_x^{\uparrow} \tag{5}$$

The notation is not sufficient here. What I mean to say is there must be a DS Rumoured or Omnipresent at all times. Does not matter which key or what algorithm.

$$A_x^{\neg\uparrow}(A_i^- \vee B_i^+ C_i^+) \vee A_x^{\uparrow} B_x^+ C_x^+ \vee A_x^+ B_x^{\uparrow} C_x^{\uparrow} A_y^+ B_y^{\downarrow} C_y^{\downarrow} \tag{6}$$

The second rule basically describes the dependency of the DS record on the DNSKEY record and the signature thereof. Either there is no DS published and the DNSKEY is well known, another DS is published with the DNSKEY well known, or two DS's are well known and their DNSKEYS are being swapped.

Although not stated explicitly, all records in the statement must have the same algorithm.

$$B_x^{\neg\uparrow}(B_i^- \vee D_i^+) \vee B_x^{\uparrow} D_x^+ \vee B_x^+ D_x^{\uparrow} B_y^+ D_y^{\downarrow} \tag{7}$$

The third rule expresses the dependency of the DNSKEY on the RRSIG. Either there is no DNSKEY at all, the DNSKEY is being introduced with the RRSIG well known, or two DNSKEYs exist and their RRSIGS's are being swapped.

Again, all record must have the same algorithm.

$$B_i \geq C_i \tag{8}$$

The state of the RRSIG DNSKEY may not be greater than the state of the DNSKEY. In the normal case both records are published together. In case of 5011 we need to introduce a new DNSKEY for some time *without* signing the DNSKEY RRset. We must however prevent the signature to be introduced or outroduced before the actual DNSKEY.

7 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.

7.1 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^-		$MaxTTL(key, sig)$
Total time			$2 \times MaxTTL(key, sig)$

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 $A^{\uparrow}_x B^+_x C^+_x$ 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.

7.2 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.