DNSSEC Key State Transitions

Yuri Schaeffer, Yuri@NLnetLabs.nl

Contents

1 Introduction . . . 1
2 Cache Centered Approach . . . 1
2.1 Key States . . . 2
2.2 Definitions . . . 3
3 DNSSEC Validity . . . 4
4 Transition Rules . . . 6
4.1 Transition rules for $Ds(k)$ . . . 6
4.2 Transition rules for $Dnskey(k)$ . . . 8
4.3 Transition rules for $Rrsig(k)$ . . . 11

1 Introduction

During a key rollover each involved key has a state. Matthijs Mekking pointed out that the statemachine for this key can be represented as three individual smaller state machines: The state at the parent, the private key state, and the public key state.

That idea is the basis for this document. A cache centered rather than a rollover centered approach is choosen and the three different state machines are generalized to one type of state machine. The three state machines now represent the public records associated with a key (DS, DNSKEY, RRSIG). The state of each record is defined by its reputation among all DNS caches in the world.

With this we are able to formalize the boundries of DNSSEC and make precise statements about the validity of a zone with respect to DNSSEC. This has two levels: One of them is that we can judge any invariant of states on validity. The other is that for each state we know under what exact circumstances we can me a transition to the next state.

2 Cache Centered Approach

The cache centered approach uses a few extra concepts. Each key has a goal, an internal desire to be either known to all caches around the world or no caches 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 compromized. 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.

Using a cache centered approach has a number of advantages. Here, in contrast to a rollover 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 rollover 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.

This point of view makes sense because the identities validating the zone are viewing it from a cache's perspective. If we make sure any possible view on the data is valid at any point in time we have done enough.

2.1 Key States

TODO: Regardless of the record type we can define 5 basic states. The distinctive property is its existence in caches world wide, so we choose a cache oriented approach. A record starts in a Generated state, G. Since the record is never published yet there is no cache having[1] this record. The next state is Submitted, S. In this state the record could be known to *any* cache, but might as well be none. As for all state the exact meaning depends on the record type, e.g. Submitted may mean emailed to parent or published in zone. State three is Propagated, P, which means all caches have the record or can obtain it. In the Withdrawn W state not all caches have the record or are able to get it. Finally Ceased C denotes no cache will try to use the record any longer.

To summarize:

G The record is not used.

S Submitted, some caches might have seen the record.

P Propagated, all caches have seen the record or will see it.

W Withdrawn, some caches are unable to get the record.

Predicate logic. The definitions above are in terms of set theory. Although entirely equivalent, it is probably useful to write the same thing in terms of predicate logic. We introduce the following boolean functions on $\mathbb{R}$, with shorthand names for reference in the remaining pages of this document.

$$\begin{array}{rcl}
H(r) & \equiv & r \in \mathit{Hidden} & = & \forall c \in \mathbb{C} : r \notin c \\
R(r) & \equiv & r \in \mathit{Rumoured} & = & (\exists c \in \mathbb{C} : r \notin c) \wedge (\exists c \in \mathbb{C} : r \in c) \wedge r \in \mathit{Announced} \\
O(r) & \equiv & r \in \mathit{Omnipresent} & = & \forall c \in \mathbb{C} : r \in c \\
S(r) & \equiv & r \in \mathit{Squashed} & = & (\exists c \in \mathbb{C} : r \notin c) \wedge (\exists c \in \mathbb{C} : r \in c) \wedge r \notin \mathit{Announced}
\end{array}$$

[1] When we say a record is in cache we explicitly mean that there are no old versions in cache with an unexpired TTL. Having to fetch a record from the authorities is also considered in cache.

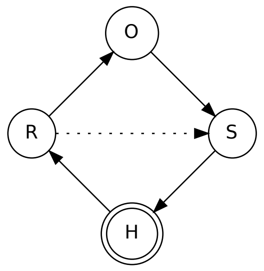

Fig. 1: State diagram for individual key components.

2.2 Definitions

Keys do not stand on their own, but actually require a number of resource records in their context. Specifically, a key and its context can be described as a tuple $\langle ds, dnskey, emphrrsig\rangle$ with the resource records for a DS, DNSKEY and RRSIG record, all for the same signing key. Define the type $K = \mathbb{R}\times\mathbb{R}\times\mathbb{R}$ represent such key with context.

Now define the following functions to capture the state H, R, O or S for each of the three components of a key with context tuple:

$$\begin{aligned}
State(r) &\equiv H, \text{ if } r \in Hidden\\
&\equiv R, \text{ if } r \in Rumoured\\
&\equiv O, \text{ if } r \in Omnipresent\\
&\equiv S, \text{ if } r \in Squashed\\
State_{\mathrm{DS}}(r,x,y) &\equiv State(r)\\
State_{\mathrm{DNSKEY}}(x,r,y) &\equiv State(r)\\
State_{\mathrm{RRSIG}}(x,y,r) &\equiv State(r)
\end{aligned}$$

- Key $k = \langle ds, dnskey, rrsig\rangle$, a tuple of states
- Key chain $\mathbb{K}$, a set of keys
- $Goal(k) \in \{O, H\}$
- $Roles(k) \subseteq \{ksk, zsk\}$
- $Alg(k) \in \mathbb{N}$, Algorithm number of key

Each zone has its own key chain (but not necessarily unique key material). A key chain is a collection of keys currently involved with the zone, regardless of their state. The key itself consists of three states, one for each record involved. It also defines a goal to which the records should try to move.

Each record is evaluated individually for the next state transition. If it not already reached its goal, it will try to move towards it provided the result is a valid DNSSEC configuration. This way only a small set of conditions is evaluated each time.

Roll overs are not explicitly defined but derived from the set of rules. As a result the system or a user could insert a new key at any time in the process. For example a user has a dual key signing scheme and sets the goal of all current keys to Ceased. Then he inserts a combined ZSK/KSK. From here the system will figure out the (save) way to having the new key introduced. It might finish pending roll overs but if possible will try to discard the old keys as soon as possible.

Setting the goal of all your keys to ceased without first introducing a new key has no implication for the validity of the zone. The system always maintains a valid configuration. If it wants to remove old keys but can not, it won't.

3 DNSSEC Validity

A consistent key(context) is defined as a key that does not introduce internal inconsistencies. A (partially) propagated KSK must have a fully propagated ZSK.

$$\begin{aligned}
&ConsistentKeys \equiv \{k | k \in \mathbb{K},\\
&\quad Roles(k) \subseteq ksk \rightarrow (\neg H(Ds(k)) \rightarrow O(Dnskey(k))) \wedge\\
&\quad \neg H(Dnskey(k)) \rightarrow O(Rrsig(k)) \wedge\\
&\quad (\\
&\quad\quad H(Dnskey(k)) \vee\\
&\quad\quad ksk = Roles(k) \rightarrow \exists k' \in \mathbb{K} \cdot (\\
&\quad\quad\quad zsk \in Roles(k') \wedge\\
&\quad\quad\quad Alg(k) = Alg(k') \wedge\\
&\quad\quad\quad O(Dnskey(k')) \wedge\\
&\quad\quad\quad O(Rrsig(k'))\\
&\quad\quad)\\
&\quad)\\
&\}
\end{aligned}$$

SafeKeys are keys that might be internally inconsistent but for which a consistent counterpart exists.

```
$SafeKeys \equiv \{k|k \in \mathbb{K},$
    $k \in ConsistentKeys \vee$
    $\forall r \in Roles(k) \cdot ($
        $\exists k' \in \mathbb{K} \cdot ($
            $Alg(k') = Alg(k) \wedge$
            $r \in Roles(k') \wedge$
            $l \in ConsistentKeys \wedge$
            $r = ksk \rightarrow (\neg H(Ds(k)) \rightarrow O(Ds(k'))) \wedge$
            $\neg H(Dnskey(k)) \rightarrow O(Dnskey(k'))$
        $)$
    $)$
$\}$
```

A zone is valid if no single key breaks validity and at least one complete chain for any algorithm exists. An insecure zone is represented by a NULL key.

```
$Valid(\mathbb{K}) \Leftrightarrow$
    $\forall k \in \mathbb{K} \cdot k \in SafeKeys \wedge$
    $\exists k \in \mathbb{K} \cdot ($
        $ksk \in Roles(k) \wedge$
        $O(Ds(k)) \wedge$
        $O(Dnskey(k)) \wedge$
        $O(Rrsig(k)) \wedge$
        $\exists k' \in \mathbb{K} \cdot ($
            $zsk \in Roles(k') \wedge$
            $O(Dnskey(k')) \wedge$
            $O(Rrsig(k')) \wedge$
            $Alg(k) = Alg(k')$
        $)$
    $)$
```

4 Transition Rules

4.1 Transition rules for $Ds(k)$

$H(Ds(k))$

If k is not a KSK the DS state does not matter. If is *is*, before submitting either DNSKEY must be propagated or an other key with the same algorithm should be ready.

$$\left.\begin{aligned} &Goal(k) = O \land ksk \notin Roles(k) \\[2ex] &Goal(k) = O \land O(Dnskey(k)) \\[2ex] &Goal(k) = O \land \exists k' \in \mathbb{K}(\\ &\qquad\qquad Alg(k) = Alg(k') \\ &\qquad\qquad \land ksk \in Roles(k') \\ &\qquad\qquad \land O(Ds(k')) \\ &\qquad\qquad \land O(Dnskey(k')) \\ &\qquad\qquad \land O(Rrsig(k'))) \end{aligned}\right\} \rightarrow [submit], R(Ds(k)) \tag{1}$$

$R(Ds(k))$

Nothing should depend on this RR yet.

$$Goal(k) = H \rightarrow S(Ds(k)) \tag{2}$$

Has the user confirmed the DS is on its way to the parent and are we sure everything propagated since then?

$$\left.\begin{aligned} &Goal(k) = O \land ksk \notin Roles(k) \\[2ex] &Goal(k) = O \land Confirmed(k_{ds}) \\ &\qquad\qquad \land T_{now} \geq T_{whatever} \end{aligned}\right\} \rightarrow O(Ds(k)) \tag{3}$$

Postpone for later.

$$\begin{aligned} Goal(k) = O &\land Confirmed(k_{ds}) \\ &\land T_{now} < T_{whatever} \\ &\rightarrow [event] \end{aligned} \tag{4}$$

$O(Ds(k))$

DS may go in State W if it refers to nothing. Or if there are other keys with the same algorithm which are valid. Or we don't break other keys and we have some other valid key(s).

$$\left.\begin{array}{rl}
Goal(k) = H \wedge \forall r \in Roles(k) & \\
& (\\
& \exists k' \in \mathbb{K}(k' \neq k \\
& \wedge Alg(k) = Alg(k') \\
& \wedge r \in Roles(k') \\
& \wedge O(Ds(k')) \\
& \wedge O(Dnskey(k')) \\
& \wedge O(Rrsig(k'))) \\
&) \\
& \\
Goal(k) = H \wedge ksk \in Roles(k) & \\
& \wedge \forall r \in Roles(k) \\
& (\\
& \forall k' \in \mathbb{K}(k' \neq k \\
& \wedge Alg(k) = Alg(k') \\
& \wedge r \in Roles(k') \\
& \rightarrow H(Ds(k'))) \\
& \wedge \\
& \exists k' \in \mathbb{K}(k' \neq k \\
& \wedge r \in Roles(k') \\
& \wedge O(Ds(k')) \\
& \wedge O(Dnskey(k')) \\
& \wedge O(Rrsig(k'))) \\
&) \\
& \\
Goal(k) = H \wedge ksk \notin Roles(k) & \\
& \wedge \exists k' \notin \mathbb{K}(k' \neq k \\
& Alg(k) = Alg(k') \\
& \wedge ksk \in Roles(k') \\
& \wedge \neg H(Dnskey(k')) \\
& \wedge O(Rrsig(k')))
\end{array}\right\} \rightarrow S(Ds(k)) \qquad (5)$$

$S(Ds(k))$

We must wait till at least $T_{whatever}$ before transition to State Ceased. As a bonus, ZSKs may transition immediately.

$$\left.\begin{array}{r} ksk \notin Roles(k) \\ \\ T_{now} \geq T_{whatever} \end{array}\right\} \rightarrow H(Ds(k)) \tag{6}$$

$$T_{now} < T_{whatever} \rightarrow [event] \tag{7}$$

4.2 Transition rules for $Dnskey(k)$

$H(Dnskey(k))$

If the signatures are propagated we may submit the Dnskey. Some other valid key is also acceptable.

$$\left.\begin{array}{rl} Goal(k) = & O \wedge O(Rrsig(k)) \\ & \wedge\, zsk \in Roles(k) \\ \\ Goal(k) = & O \wedge O(Rrsig(k)) \\ & \wedge\, \exists k' \in \mathbb{K}(\\ & Alg(k') = Alg(k) \\ & \wedge\, zsk \in Roles(k') \\ & \wedge\, O(Ds(k')) \\ & \wedge\, O(Dnskey(k')) \\ & \wedge\, O(Rrsig(k'))) \\ \\ Goal(k) = & O \wedge \forall r \in Roles(k) \\ & (\\ & \exists k' \in \mathbb{K}(k' \neq k \\ & \wedge\, Alg(k') = Alg(k) \\ & \wedge\, r \in Roles(k') \\ & \wedge\, O(Ds(k')) \\ & \wedge\, O(Dnskey(k')) \\ & \wedge\, O(Rrsig(k'))) \\ &) \end{array}\right\} \rightarrow R(Dnskey(k)) \tag{8}$$

$R(Dnskey(k))$

Skip the propagated state. Nobody heard of it anyway.

$$Goal(k) = H \rightarrow S(Dnskey(k)) \tag{9}$$

We did wait long enough to make sure the dnskey record is known in every cache?

$$Goal(k) = O \wedge T_{now} \geq T_{whatever} \rightarrow O(Dnskey(k)) \tag{10}$$

We did not wait long enough: Wait some more.

$$Goal(k) = O \wedge T_{now} < T_{whatever} \rightarrow event(T_{whatever}) \tag{11}$$

$O(Dnskey(k))$

If not part of a chain, withdraw. If there is still a DS make sure there is some other valid chain for this algorithm. If none for this algorithm are broken, some other algorithm will do as well.

$$\left.\begin{aligned}
&Goal(k) = H \wedge H(Ds(k)) \\
&\qquad\qquad \wedge H(Rrsig(k)) \\
\\
&Goal(k) = H \wedge \forall r \in Roles(k) \\
&\qquad\qquad (\\
&\qquad\qquad \exists k' \in \mathbb{K}(k' \neq k \\
&\qquad\qquad \wedge Alg(k') = Alg(k) \\
&\qquad\qquad \wedge r \in Roles(k') \\
&\qquad\qquad \wedge O(Ds(k')) \\
&\qquad\qquad \wedge O(Dnskey(k')) \\
&\qquad\qquad \wedge O(Rrsig(k'))) \\
&\qquad\qquad) \\
\\
&Goal(k) = H \wedge H(Ds(k)) \\
&\qquad\qquad \wedge \forall r \in Roles(k) \\
&\qquad\qquad (\\
&\qquad\qquad \forall k' \in \mathbb{K}(k' \neq k \\
&\qquad\qquad \wedge Alg(k') = Alg(k) \\
&\qquad\qquad \wedge r \in Roles(k') \\
&\qquad\qquad \rightarrow H(Dnskey(k'))) \\
&\qquad\qquad \wedge \\
&\qquad\qquad \exists k' \in \mathbb{K}(k' \neq k \\
&\qquad\qquad \wedge r \in Roles(k') \\
&\qquad\qquad \wedge O(Ds(k')) \\
&\qquad\qquad \wedge O(Dnskey(k')) \\
&\qquad\qquad \wedge O(Rrsig(k'))) \\
&\qquad\qquad)
\end{aligned}\right\} \rightarrow S(Dnskey(k)) \qquad (12)$$

$S(Dnskey(k))$

State may transition to Ceased given enough time passed to propagate change.

$$\left.\begin{array}{r} ksk \notin Roles(k) \\ \\ T_{now} \geq T_{whatever} \end{array}\right\} \rightarrow H(Dnskey(k)) \tag{13}$$

Else, schedule event.

$$T_{now} < T_{whatever} \rightarrow event(T_{whatever}) \tag{14}$$

4.3 Transition rules for $Rrsig(k)$

$H(Rrsig(k))$

Signatures for ZSKs can be introduced. For KSKs there must be a valid ZSK.

$$Goal(k) = O \rightarrow R(Rrsig(k)) \tag{15}$$

$R(Rrsig(k))$

Don't wait till signatures are propagated.

$$Goal(k) = H \rightarrow S(Rrsig(k)) \tag{16}$$

Enough time passed to propagate signatures. Or if we do a smooth transition

$$\left.\begin{array}{l} Goal(k) = O \wedge T_{now} \geq T_{whatever} \\ \\ Goal(k) = O \wedge AllowSmooth \\ \qquad\qquad \wedge \forall r \in Roles(k)(\\ \qquad\qquad \exists k' \in \mathbb{K}(\\ \qquad\qquad Alg(k') = Alg(k) \\ \qquad\qquad \wedge r \in Roles(k') \\ \qquad\qquad \wedge O(Dnskey(k')) \\ \qquad\qquad \wedge O(Rrsig(k')) \\ \qquad\quad)) \end{array}\right\} \rightarrow O(Rrsig(k)) \tag{17}$$

Not enough time passed to propagate signatures.

$$Goal(k) = O \wedge T_{now} < T_{whatever} \rightarrow event(T_{whatever}) \tag{18}$$

$O(Rrsig(k))$

If the dnskey is gone from all caches can we withdraw the signatures. We may break the chain if other valid keys are available.

$$\left.\begin{array}{l} Goal(k) = H \wedge H(Dnskey(k)) \\ \\ Goal(k) = H \wedge \forall r \in Roles(k) \\ \qquad (\\ \qquad\ \exists k' \in \mathbb{K}(k' \neq k \\ \qquad\ \wedge Alg(k) = Alg(k') \\ \qquad\ \wedge r \in Roles(k') \\ \qquad\ \wedge O(Ds(k')) \\ \qquad\ \wedge O(Dnskey(k')) \\ \qquad\ \wedge O(Rrsig(k'))) \\ \qquad) \end{array}\right\} \rightarrow S(Rrsig(k)) \tag{19}$$

$S(Rrsig(k))$

Dnskey is gone so no one needs these signatures.

$$\left.\begin{array}{r} H(Dnskey(k)) \\ \\ T_{now} \geq T_{whatever} \end{array}\right\} \rightarrow H(Rrsig(k)) \tag{20}$$

$$T_{now} < T_{whatever} \rightarrow event(T_{whatever}) \tag{21}$$